



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

DISSERTATION

**TRI-LEVEL OPTIMIZATION ALGORITHMS FOR
SOLVING DEFENDER-ATTACKER-DEFENDER
NETWORK MODELS**

by

Gary L. Lazzaro

June 2016

Dissertation Supervisor:

W. Matthew Carlyle

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE June 2016	3. REPORT TYPE AND DATES COVERED Dissertation		
4. TITLE AND SUBTITLE TRI-LEVEL OPTIMIZATION ALGORITHMS FOR SOLVING DEFENDER- ATTACKER-DEFENDER NETWORK MODELS			5. FUNDING NUMBERS	
6. AUTHOR(S) Gary L. Lazzaro				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____ N/A ____.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) The optimal defense and operation of networks against worst-case attack is an important problem for military analysts. We review development of existing solutions for the Defender-Attacker-Defender (DAD) tri-level optimization model and investigate new applications and solution procedures. We develop an implicit enumeration algorithm that incorporates addition of new defenses as an alternative solution method for the DAD model. Our testing demonstrates that implicit enumeration can efficiently generate all equivalent optimal or near-optimal solutions for DAD problems. When budgets for network defense or attack are uncertain, decision makers usually prioritize defenses in nested lists. We quantify the costs of various strategies for nesting of defenses. We design a parametric programming formulation of the DAD model to find nested defenses that have the smallest cost difference from optimal non-nested solutions. We create new solution procedures for the DAD constrained shortest path problem. We merge the attacker model with Lagrangian relaxation of the operator model into a single formulation that can obtain fast heuristic solutions. We combine our heuristic algorithm with traditional methods to obtain provably optimal or near-optimal solutions. We test our algorithms on medium and large networks, and our results show that our innovations can significantly outperform traditional nested decomposition.				
14. SUBJECT TERMS Defender-attacker-defender, network optimization, tri-level optimization, implicit enumeration, nested defense, prioritized lists, constrained shortest path, minimum cost flow, algorithms, network defense, network interdiction, network modeling, system defense			15. NUMBER OF PAGES 259	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**TRI-LEVEL OPTIMIZATION ALGORITHMS FOR SOLVING DEFENDER-
ATTACKER-DEFENDER NETWORK MODELS**

Gary L. Lazzaro
Commander, United States Navy
B.S., University of California, San Diego, 1996
M.S., Naval Postgraduate School, 2013

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN OPERATIONS RESEARCH

from the

**NAVAL POSTGRADUATE SCHOOL
June 2016**

Approved by: W. Matthew Carlyle
Professor of
Operations Research
Dissertation Supervisor

Gerald Brown
Distinguished Professor of
Operations Research

Michael Atkinson
Associate Professor of
Operations Research

Emily Craparo
Assistant Professor of
Operations Research

Craig Rasmussen
Professor and Chair, Department of Applied Mathematics

Approved by: Patricia Jacobs, Chair, Department of Operations Research

Approved by: Douglas Moses, Vice Provost of Academic Affairs

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The optimal defense and operation of networks against worst-case attack is an important problem for military analysts. We review development of existing solutions for the Defender-Attacker-Defender (DAD) tri-level optimization model and investigate new applications and solution procedures. We develop an implicit enumeration algorithm that incorporates addition of new defenses as an alternative solution method for the DAD model. Our testing demonstrates that implicit enumeration can efficiently generate all equivalent optimal or near-optimal solutions for DAD problems. When budgets for network defense or attack are uncertain, decision makers usually prioritize defenses in nested lists. We quantify the costs of various strategies for nesting of defenses. We design a parametric programming formulation of the DAD model to find nested defenses that have the smallest cost difference from optimal non-nested solutions. We create new solution procedures for the DAD constrained shortest path problem. We merge the attacker model with Lagrangian relaxation of the operator model into a single formulation that can obtain fast heuristic solutions. We combine our heuristic algorithm with traditional methods to obtain provably optimal or near-optimal solutions. We test our algorithms on medium and large networks, and our results show that our innovations can significantly outperform traditional nested decomposition.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	DEFENDER-ATTACKER-DEFENDER MODEL	1
A.	PROBLEM DEFINITION	1
B.	HISTORICAL BACKGROUND.....	2
C.	DAD NESTED DECOMPOSITION ALGORITHMS	7
1.	DAD Minimum Cost Flow Model Formulation	7
2.	AD Inner Decomposition.....	10
3.	DAD Outer Decomposition	14
D.	EXAMPLE APPLICATION OF DAD PROBLEM	16
E.	SUMMARY OF CONTRIBUTIONS.....	18
1.	Efficient Enumeration Algorithms for the DAD Model.....	18
2.	Impact of Nested Defenses on Optimality	19
3.	Incorporating Heuristics to Speed Up DAD Solution Algorithms	21
II.	IMPLICIT ENUMERATION OF DAD PROBLEM	23
A.	IMPLICIT ENUMERATION THEORY	23
1.	Enumeration of Defenses in the Literature	24
2.	Contributions to IE Theory.....	25
B.	ENUMERATION TREE CONCEPTS	27
1.	Data Structures at an IE Tree Node.....	28
2.	Requirements of IE tree branches.....	29
C.	IE ALGORITHM FUNDAMENTALS.....	30
1.	Skipping and Reconsideration of Potential Defenses	31
2.	Avoiding Repeated Calculations.....	32
D.	PRUNING OF NEW EDGES	34
1.	A Simple Pruning Rule for New Edges Is Incorrect.....	35
2.	Excess Supply and Incentive	39
3.	New-Edge Pruning Rule	43
4.	Examples of New Edge Pruning	46
E.	UPPER AND LOWER BOUNDS IN IE.....	49
F.	IMPLICIT ENUMERATION ALGORITHM.....	52
1.	Depth First Search Algorithm Option	54
2.	Algorithm Testing and Performance	54
G.	ENUMERATION SUMMARY AND CONCLUSIONS	57
III.	NESTED DEFENSES IN THE DAD PROBLEM.....	61
A.	INTRODUCTION.....	61

B.	IMPLEMENTATION OF NESTED DEFENSES	64
1.	Nesting as Constraint Equations	65
2.	Nesting by Fixing Variables	65
C.	NESTED DEFENSE HEURISTIC APPROACH	67
1.	Heuristic Starting Point Assumption	69
2.	Parameterized Defense Budget and Known Attack Budget	70
3.	Known Defense Budget and Parameterized Attack Budget	73
4.	Parameterized Defense and Attack Budgets	78
D.	NESTED DEFENSE ALGORITHM STEP SIZE	87
E.	NESTED DEFENSE DISTANCE FROM OPTIMALITY MEASUREMENT	92
1.	Vector Analysis	92
2.	Matrix Analysis	94
3.	Percentage Gap From Optimality	98
4.	Minimax Regret Analysis	99
5.	Equal Likelihood Analysis	100
F.	NESTED DEFENSES WITH A SUBOPTIMAL STARTING SCENARIO	101
G.	OPTIMIZING THE BEST NESTED DEFENSE	105
1.	Best Nested Defense as Measured by 1-Norm	106
2.	Best Nested Defense as Measured by Infinity-Norm	111
3.	Best Nested Defense as Measured by 2-Norm	113
4.	Implementation on Test Network	114
H.	NESTED DEFENSES IN MAXIMUM FLOW PROBLEMS	122
I.	NESTED DEFENSE SUMMARY	128
IV.	DAD CONSTRAINED SHORTEST PATH PROBLEM	131
A.	PROBLEM DEFINITION	131
1.	Introduction	131
2.	Problem Formulation	136
3.	Regular Nested Decomposition Approach	139
B.	ARTIFICIAL TEST NETWORKS	143
C.	LAGRANGIAN RELAXATION OF OPERATOR MODEL	150
D.	IMPROVEMENTS TO THE INNER DECOMPOSITION ALGORITHM	154
1.	Lagrange Variable Problem Transformation	154
2.	Path Enumeration Closes Lagrangian Relaxation Optimality Gap	157
3.	Multi-cut Inputs to AD Inner Master Problem	158
E.	RELAXATION IN INNER AND OUTER DECOMPOSITION	162

F.	TWO HEURISTIC ALGORITHMS FOR DAD CSP.....	165
1.	DAD CSP Phase 1 Bounding Heuristic Algorithm	166
2.	DAD CSP Phase 1 Speed Heuristic Algorithm.....	169
G.	EXTENDING THE HEURISTIC TO OBTAIN PROVABLY OPTIMAL SOLUTIONS	171
H.	RESULTS ON ARTIFICIAL TEST NETWORK.....	174
1.	Results for Combination of the Bounding and Optimality Algorithms	178
2.	Speed and Optimality Algorithms vs. Regular Decomposition	182
I.	APPLICATION OF DAD CSP TO A REAL WORLD NETWORK	191
1.	Definition of Real World Network	191
2.	DAD CSP Example Scenario on Real-World Network	196
3.	Algorithm Performance Comparison on Real-World Network.....	198
4.	Lessons Learned from Real-World Network	212
J.	SUMMARY	213
V.	CONCLUSIONS AND FUTURE WORK	217
A.	SUMMARY OF FINDINGS	217
B.	FUTURE RESEARCH TOPICS	221
1.	Implicit Enumeration for DAD Constrained Shortest Paths	222
2.	Implicit Enumeration Application to Larger or Difficult Problems	222
3.	Attack Effects on Side Constraints.....	223
4.	Determination of Lagrange Multiplier for Outer Decomposition	223
5.	DAD Modeling for Other Kinds of Systems	223
	APPENDIX. ADDITIONAL ALGORITHM PSEUDO-CODE	225
A.	RECURSIVE PATH ENUMERATION ALGORITHM	225
B.	AD CSP WITH LAGRANGIAN RELAXATION AND PATH ENUMERATION ALGORITHM.....	226
	LIST OF REFERENCES	227
	INITIAL DISTRIBUTION LIST	233

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1.	Test Network for DAD Models. Adapted from Alderson et al. (2015).....	16
Figure 2.	Optimal Defenses Change as the Attack Budget Changes. Source: Alderson et al. (2015).	20
Figure 3.	Optimal Defense with Addition of Edges on Test Network	27
Figure 4.	Portion of an Implicit Enumeration Tree	31
Figure 5.	The Potential for Repeated Calculations in an IE Tree.....	33
Figure 6.	Exclusion Lists in an IE Tree.....	34
Figure 7.	A Minimum Cost Flow Network with a Transshipment Node.....	36
Figure 8.	New Edge Consideration with Transshipment Nodes	37
Figure 9.	A Minimum Cost Flow Network with No Excess Supply.....	38
Figure 10.	Lack of Supply in Excess of Total Demand	39
Figure 11.	No Incentive Exists for New Edge to Demand Node	41
Figure 12.	Disconnected Subnetworks with Unbuilt Connecting Edges	41
Figure 13.	Equivalent Effect of Excess Supply Source Nodes	45
Figure 14.	Attack with Excess Supply Source Nodes	45
Figure 15.	New Edge Unreachable by Excess Supply Source Nodes.....	46
Figure 16.	Worst-case Attack with Excess Supply Nodes	47
Figure 17.	Pruning New Edges in an IE Tree.....	49
Figure 18.	Upper and Lower Bounds in an IE Tree	51
Figure 19.	IE Tree with Multiple Optimal Solutions	57
Figure 20.	Three Dimensional Plots of Test Network Optimal Solutions	64
Figure 21.	Nesting with Parameterized Defense Budgets at Various Start Points.....	71
Figure 22.	Equivalent Persistent Defenses on the Test Network	75

Figure 23.	Persistent Defenses with Parameterized Attack Budgets.....	77
Figure 24.	Nesting Strategies for Parameterized Defense and Attack Budgets	83
Figure 25.	Nesting with Parameterized Defense and Attack Budgets	86
Figure 26.	Effect of Budget Step Size on Nested Defense.....	89
Figure 27.	Nested Defenses for a Two Unit Step Size.....	90
Figure 28.	Definitions of Vector Norms. Adapted from Leon (2010).	93
Figure 29.	Description of Matrix Norms. Adapted from Leon (2010).....	95
Figure 30.	Use of the Matrix 1-Norm for Unknown Defense Budgets.....	96
Figure 31.	Use of the Matrix Infinity-Norm for Unknown Attack Budgets	96
Figure 32.	Difference Matrices for Unknown Defense and Attack Budgets	97
Figure 33.	Optimality Gap Equation. Adapted from Koc and Morton (2015).....	98
Figure 34.	Objective Function Values for Various Nesting Strategies	103
Figure 35.	Difference Vectors for Suboptimal Nesting Example	104
Figure 36.	Test Network Nested Defense with 1-Norm Nesting Example	116
Figure 37.	Difference Vectors for 1-Norm Nesting Example	116
Figure 38.	Test Network Nested Defense with Infinity-Norm Nesting Example	119
Figure 39.	Difference Vectors for Infinity-Norm Nesting Example	119
Figure 40.	Test Network Nested Defense with Squared 2 Norm Objective	121
Figure 41.	Difference Vectors for 2-Norm Nesting Example	122
Figure 42.	Maximum Flow Network. Adapted from Alderson et al. (2013)	125
Figure 43.	Maximum Flow Network with Defense Budget 2, Attack Budget 5.....	126
Figure 44.	Six Node Test Network. Adapted from Ahuja et al. (1993, p. 599)	144
Figure 45.	50 Node Test Network	145
Figure 46.	Shortest Cost and Shortest Time Paths for 50 Node Test Network	147
Figure 47.	Nested Decomposition Solution Times in 50 Node Test Network.....	149

Figure 48.	Lagrangian Relaxation and Path Enumeration on Test Network.....	160
Figure 49.	Multi-cut Algorithm Improvement on Six Node Test Network	161
Figure 50.	Optimal Defense, Attack and Shortest Path for DAD CSP Example	176
Figure 51.	Relative Comparison of Solution Speed on 50 Node Test Network	185
Figure 52.	Relative Comparison of Iterations on 50 Node Test Network.....	191
Figure 53.	MD-VA-DC Road Network. Adapted from Carlyle and Wood (2005). .	192
Figure 54.	Road Network DC area. Adapted from Carlyle and Wood (2005).....	193
Figure 55.	Shortest Distance and Time Paths from the Pentagon to Appomattox	194
Figure 56.	Map from the Pentagon to Appomattox. Source: Google Maps (2016).....	195
Figure 57.	Example Problem for Optimal Defense and Attack of Road Network....	197
Figure 58.	Solution Time Relative Performance for Road Network.....	207
Figure 59.	Relative Difference of Iterations Performed for Road Network.....	212

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1.	Test Network Solution with Fixed Defense Budget of Four Units.....	17
Table 2.	Test Network Solution with Fixed Attack Budget of Four Units	18
Table 3.	Comparison of Solution Times for Different DAD Solution Methods.....	55
Table 4.	Effect of Defense Budget Starting Point on Nested Defense Solution	68
Table 5.	Effect of Defense Budget Starting Point on Nested Defenses Chosen	69
Table 6.	Effect of Attack Budget Starting Point on Persistent Defenses	74
Table 7.	Effect of Attack Budget Starting Point on Persistent Defenses	74
Table 8.	Nested Defense Obtained by Varying Defense Budget.....	80
Table 9.	Nested Defense Obtained by Varying Attack Budget	80
Table 10.	Vector Norm Analysis at Different Starting Point Instances.....	94
Table 11.	Matrix Norm Analysis at Different Starting Point Instances.....	97
Table 12.	Test Network Optimality Gap Percentages.....	99
Table 13.	Decision Matrix for Unknown Defense and Attack Budgets	100
Table 14.	Regret Matrix for Unknown Defense and Attack Budgets	100
Table 15.	Equal Likelihood Analysis for Unknown Defense and Attack Budgets..	101
Table 16.	Objective Function Values for Suboptimal Nesting Example.....	102
Table 17.	Nested Defenses in Test Network for Suboptimal Nesting Example	103
Table 18.	Difference Norms for Suboptimal Nesting Example.....	104
Table 19.	Objective Function Values in 1-Norm Nesting Example	115
Table 20.	Nested Defenses Chosen in 1-Norm Nesting Example	115
Table 21.	Difference Norms for 1-Norm Nesting Example.....	117
Table 22.	Objective Function Values for Infinity-Norm Nesting Example.....	117
Table 23.	Nested Defenses Chosen for Infinity-Norm Nesting Example.....	118

Table 24.	Difference Norms for Infinity-Norm Nesting Example.....	119
Table 25.	Objective Function Values for Squared 2-Norm Nesting Example.....	120
Table 26.	Nested Defenses Chosen for Squared 2-Norm Nesting Example.....	120
Table 27.	Difference Norms for 2-Norm Nesting Example.....	122
Table 28.	Optimal Defenses and Attacks in Maximum Flow Network Example....	127
Table 29.	Arc Cost and Time Values for 50 Node Test Network.....	146
Table 30.	Lagrangian Relaxation and Path Enumeration on Test Network.....	160
Table 31.	Multi-cut Algorithm Improvement on Six Node Test Network	161
Table 32.	Optimal Defense, Attack and Shortest Path for DAD CSP Example	175
Table 33.	Upper and Lower Bounds from Phase 1 Bounding Heuristic.....	179
Table 34.	Solution Times for Phase 1 and 2 vs. Nested Decomposition	183
Table 35.	Iteration Comparison of Phase 1 and 2 vs. Nested Decomposition.....	187
Table 36.	Shortest Distance and Time Paths from the Pentagon to Appomattox....	195
Table 37.	Relative Optimality Tolerances for DAD CSP	200
Table 38.	Real World Network Solution Values and Relative Optimality Gap	202
Table 39.	Solution Time Performance Comparison on Real World Network	205
Table 40.	Comparison of DAD CSP Iterations for Real World Network.....	209

LIST OF ACRONYMS AND ABBREVIATIONS

AD	Attacker-Defender Model
CPU	Central Processing Unit
CSP	Constrained Shortest Path
DA	Defender-Attacker Model
DC	Washington, District of Columbia
DD	Defender-Defender Model
DAD	Defender-Attacker-Defender Model
GAMS	General Algebraic Modeling Software
IBM	International Business Machines
ICBM	Intercontinental Ballistic Missile
IE	Implicit Enumeration
ILP	Integer Linear Program
LP	Linear Program
MD	Maryland
MIP	Mixed Integer Program
NP	Non-deterministic Polynomial Time Algorithm
RELTOL	Relative Optimality Tolerance
RIMF	R-Interdiction Median with Fortification problem
SEC	Solution Elimination Constraint
SP	Shortest Path Problem
VA	Virginia

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

The defender-attacker-defender (DAD) problem is a tri-level optimization model of the defense and operation of a system against a worst-case enemy attack. We model the DAD problem as a sequential game. First, the defender prepares for an attack by either hardening portions of a system or, in some cases, designing new additions to a system. Second, an attacker seeks to degrade system performance despite the defensive preparations. Finally, the defender of the system makes the best possible decisions to operate the resulting system. The solution to a DAD problem is a defense of the system to make it more resilient to a worst-case attack. Expression 1 is a simplified representation of DAD tri-level optimization model with decision variables for defense (W), attack (X) and operation (Y) of a system. We describe the historical development of the DAD model and our research develops three concepts for the DAD model.

$$\min_{W \in \Psi} \max_{X \in \Xi} \min_{Y \in \Upsilon(W)} f(W, X, Y) \quad (1)$$

We advance an alternative solution algorithm for the DAD model with implicit enumeration (IE). IE of the DAD model works by restricting and subsequently relaxing the defense budget of the original problem instance. IE reduces the tri-level DAD problem into a progression of simpler bi-level subproblems arranged in a tree structure. We expand enumeration theory by including the ability to design new additions to a network as defenses against attack. We note that IE is an alternative algorithm to obtain solutions to DAD problem instances because it does not use nested decomposition.

We determine that IE is usually slower than traditional nested decomposition to find an optimal solution to a DAD model. However, in problem instances where multiple equivalent optimal solutions exist, IE can find all of the equivalent solutions concurrently. Nested decomposition does not have such a capability. Furthermore, in situations where all solutions within a specific percentage of optimality are required, IE can generate all possible solutions simultaneously. We can conclude that IE is an efficient way to generate many equivalent solutions to a DAD minimum cost flow problem instance.

We investigate the implications of producing a prioritized list of defenses for a system when the defense or attack budget is unknown. We use the term *nested defenses* to refer to a monotonic sequence of sets, where each set of defenses for a particular budget scenario contains the defenses for the next smaller budget scenario. We examine how nested defenses differ from the optimal set of defenses for any particular defense and attack budget scenario for the DAD minimum cost flow problem. In particular, we quantify the cost of requiring a set of defenses to be monotonic. We examine various heuristic nesting strategies and compare their performance against known optimal solutions. We adapt the use of vector and matrix norms as well as various decision theory tools to measure how a set of nested defenses differs from the set of optimal defenses. We define the *best* nested defense to be the set of defenses that is closest to the optimal set of non-nested defenses in terms of a vector norm measurement over the set of unknown budget scenarios. We demonstrate how heuristic approaches may not produce the best set of nested defenses.

In order to find the best nested defense, we develop a parametric programming formulation of the DAD model that allows for unknown defense or attack budgets. This formulation minimizes the overall system cost by considering all unknown budget scenarios simultaneously. Our formulation fits within the framework of stochastic programming. We can conclude that our parametric programming formulation can generate the set of nested defenses that is as close as possible to the set of optimal solutions over all possible budget scenarios.

We develop a DAD model for the constrained shortest path (CSP) problem. We do not allow for new additions as defenses in this model. To the best of our knowledge, solution procedures for DAD CSP do not exist in the literature. The CSP problem is arduous because the additional side constraint breaks the network structure of the problem. Loss of network structure results in a much more difficult binary linear program to solve. We adapt nested decomposition techniques to solve DAD CSP instances on test networks. We find that the nested decomposition algorithm by itself for DAD CSP problem instances becomes either intractable or excessively time intensive to solve as defense and attack budgets increase.

We improve the ability to solve a DAD CSP instance by developing a Lagrangian relaxation heuristic. We expand known Lagrangian relaxation techniques to include defense and attack of the CSP problem. We develop two different alternative algorithms for Lagrangian relaxation of DAD CSP. One algorithm uses path enumeration techniques to close the gap between the bounds that are created by Lagrangian relaxation. The second algorithm takes advantage of the similar maximization goals of Lagrangian relaxation and the attacker model. We merge the attacker model and the Lagrangian relaxation operator model into a single model. Both algorithms find a fast heuristic solution to DAD CSP.

We combine our heuristics with nested decomposition in order to reduce the time required to solve a DAD CSP instance optimally. Our heuristic algorithms cannot be guaranteed to give an optimal solution, but they can provide information to accelerate nested decomposition. We use output from our heuristic algorithms as starting values for nested decomposition. Our combined algorithms are significantly faster than regular nested decomposition to find an optimal or near optimal solution to a DAD CSP instance.

We compare results for our combined algorithms against regular nested decomposition for DAD CSP on a randomly generated 50 node grid network as well as a large road network of the Washington, DC, area. Except for a few test cases, our combined algorithm is able to find the known optimal solution much faster than regular nested decomposition on the grid network. In some instances, we observe that our combined algorithm can be more than four times faster than regular nested decomposition in finding the optimal solution on the grid network. The road network shows that our DAD CSP algorithms can only scale to large problem sizes with approximately optimal solutions. In order to obtain a solution to a DAD CSP instance on the road network in less than 24 hours, we allow for solutions that are proven to be within 10% of optimality. Our combined algorithm is able to find an approximately optimal solution faster than regular nested decomposition in 19 of 25 budget scenarios on the real-world road network. In the best case, our combined algorithm can be 66 times faster than regular nested decomposition for DAD CSP on the road network. We conclude that our combined algorithms represent a significant improvement in the ability to solve DAD CSP instances.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

This dissertation has my name on it, but many talented people supported me throughout the process of learning, researching, and writing.

I am deeply indebted to my advisor, Professor Matt Carlyle, for his patience, support, resourcefulness, innovation, and creative mind. You were the first professor that I met as a Ph.D. student and you stayed alongside me every step of the journey. You took the time to mentor me in all of the finer points of both Operations Research as well as the differences between a master's degree student and a doctoral candidate. You helped me to remain focused and pick up the pieces whenever setbacks occurred along the way. Thank you for the countless hours we spent in your office over the past three years as you molded me to think on a philosophical level and figure out why things work.

Thank you to all of the members of my dissertation committee: Distinguished Professor Gerald Brown, Professor Craig Rasmussen, Associate Professor Michael Atkinson, and Assistant Professor Emily Craparo. I am grateful for all of the assistance you provided me in preparation for my Ph.D. qualification examinations and for all of the insights that you provided to make this dissertation the best it can be.

I would also like to thank other professors that helped me to enjoy Operations Research: Professor Kyle Lin, for the time you took with me to help me to learn how and why stochastic models work and Professors Robert Koyak and Tom Lucas, for all of the extra effort that you invested in me to understand statistics at a very deep level. Also, thank you Professor Ned Dimitrov for providing the initial spark that made me very curious about network modeling.

Thank you to all of my fellow Operations Research PhD students Christian Klaus, Sofia Miranda, Jesse Pietz, Dick McGrath, Matt Hawks and "Austin" Yu-Chia Wang. I also have to mention Jon Roginski in Applied Mathematics. I am the last person remaining in the Ph.D. student office, and it is very challenging to complete doctoral research alone. All of your encouragement and help during the early stages of being a Ph.D. student were of immense help to me as I cross the finish line by myself. I doubt I could have succeeded

in this program without the help of my friends. I would also like to mention my friends in the master's degree cohort that graduated in 2015, especially Dustin Shultz, Brian Colburn and the many other students for making our time in class and group projects (and head scratching) truly enjoyable. Someone once said "you never know the friends you have until you need to move," and I know that you answered my call.

To my family, thank you for guiding me to make good decisions throughout my life. Dad, I never would have been a naval aviator without your encouragement to follow your lead and join the military. Mom, thank you for your encouragement to achieve excellence in academics. I know that a Ph.D. in Operations Research is far from a first-grade classroom, but a desire to be the best at everything you try is the same everywhere. To my brother, Terry, I appreciate you for being there for me over the past 40 years.

To my son, Anthony, my daughter, Rebecca, and our newborn baby, Vanessa, each one of you is the reward for all of this hard work. I dedicate all of my research to you!

I am immeasurably beholden to my wife, Kathryn. My gratitude alone could fill this dissertation, but I will keep it to a paragraph. Overworked and underappreciated, you supported me throughout the many ups and downs of my naval career. You were the first person to believe in my decision to become a permanent military professor and get a Ph.D. You remained encouraging through all of my mistakes, stubbornness, and stumbling progress toward this degree. I certainly will not forget the sacrifices you made during all of the long nights, weekends, holidays, missed vacations, and other random times that I had to be at the office to figure out how to do something. I am very proud to have such a perfect person to be my wife and the mother of our wonderful children.

I. DEFENDER-ATTACKER-DEFENDER MODEL

A. PROBLEM DEFINITION

This dissertation investigates applications and solutions of defender-attacker-defender (DAD) tri-level models to a variety of system optimization problems. Our goals are to expand existing knowledge about the problem structure and improve solution procedures for it. Alderson, Brown, Carlyle, and Wood (2011) define a DAD system optimization problem as a tri-level optimization model that models the optimal defense and operation of a friendly system against a worst-case enemy attack. We model a DAD problem as a sequential game in which the decisions are made in a predetermined sequence: the defender, then the attacker and finally the defender (or operator) (Brown et al., 2005). First, the defender of a system optimally prepares for a worst-case enemy attack by designing new additions to, or redesigning some parts of, a network or infrastructure system. Second, an enemy attacker seeks to optimally degrade the overall performance of the system despite all defensive preparations that have been made. Finally, the defender (or operator) of the system makes the best possible decisions to operate the resulting system optimally. The system defender and operator could be synonymous or they could be different entities with similar interests. The solution to a DAD problem is a defense or redesign of the system to make it more resilient to a set of possible attacks by explicitly considering the system's response to each attack. Alderson, Brown and Carlyle (2014) define the "operational resilience" of a system is dependent upon its set of responses and the corresponding total costs to defenses and attacks.

The DAD model can be formulated for any operator model, to include linear, non-linear, integer and binary programs. A common application for the DAD model is a network flow problem. As an example, we formally define the tri-level DAD optimization model in the context of a minimum cost network flow problem. We define a network $G = (N, A)$ with supplies of resources at source nodes, demands for resources at destination nodes, and arcs with per unit transportation costs (c_{ij}) and maximum flow capacities (u_{ij}). The classical minimum cost flow problem moves all resources from sources to destinations at the lowest cost. The amount of resources that flow on arcs is

governed by non-negative integer variables (Y). This is the simplest version of the *operator problem* (1.1a). If an attacker can add a penalty (q_{ij}) to some arcs so that the adjusted cost of an attacked arc (X) becomes $c_{ij} + q_{ij}$, we obtain the *attacker problem* (1.1b). If we allow a defender the ability to protect a number of arcs from an attack (W), we obtain the *defender problem* (1.1c). Expressions (1.1a)-(1.1c) all have blocks of constraints that govern the employment of defenses (Ψ), attacks (Ξ) and operation of flows subject to a particular defense ($Y(W)$). These blocks of constraints are detailed in later sections of this chapter.

$$\min_{Y \in \Upsilon} f(Y) \tag{1.1a}$$

$$\max_{X \in \Xi} \min_{Y \in \Upsilon} f(X, Y) \tag{1.1b}$$

$$\min_{W \in \Psi} \max_{X \in \Xi} \min_{Y \in Y(W)} f(W, X, Y) \tag{1.1c}$$

Expression (1.1c) is a DAD model for minimum cost flow and it may be solved with a nested decomposition algorithm. Israeli uses the term nested decomposition to decompose a tri-level optimization problem to simpler single and bi-level sub problems. (1999). The key insight of nested decomposition is that the inner stage of decomposition functions as a sub-problem for an outer stage of decomposition. If we employ decomposition to solve (1.1c), and use the defenses (W) as the outer decomposition master problem variables, the resulting sub problem is a two-stage model along the lines of (1.1b), which itself can be solved using decomposition. In a nested decomposition approach, the values of some variables are held constant while the remaining variables are optimized. We define the hat symbol ($\hat{\cdot}$) to denote the holding of a decision variable to a fixed value during various stages of decomposition.

B. HISTORICAL BACKGROUND

The composition of the DAD problem involves three levels of optimization, and can best be described by considering each part of the tri-level optimization sequentially (Alderson et al., 2014). The innermost level of optimization only involves the operation

of the system as originally designed, under fixed values of defenses (W) and attacks (X). The optimal operation of a system is referred to as the operator model presented as (1.2a). This model could be a network flow problem, but DAD models can include any model of an infrastructure system as the innermost model. Existing operations research and engineering literature abounds with operator models of various infrastructure systems. Ford and Fulkerson (1962) originally considered operator models in the context of maximizing the amount of goods moved from a source to a receiver.

$$\min_{Y \in Y(\hat{W})} f(\hat{W}, \hat{X}, Y) \quad (1.2a)$$

The second level of optimization in a DAD model involves the attacker, who wants to optimally choose the best set of attacks, under a fixed value of defenses (W). The goal of the attacker is to maximize the disruption to the operator of the system. Wollmer (1964) considered removing edges in a network in order to disrupt and reduce the amount of flow. The attacker model is shown in (1.2b) and is also referred to as the attacker-defender (AD) problem.

$$\max_{X \in \Xi} \min_{Y \in Y(\hat{W})} f(\hat{W}, X, Y) \quad (1.2b)$$

The problem of optimal attack and subsequent operation of a system is usually formulated as a bi-level optimization problem. The first known application of this problem comes from declassified research of Harris and Ross (1955). The authors studied the least costly methods to block the flows of a railway network moving war supplies from the Soviet Union into Europe with a finite number of attacks. Some other examples of the kinds of infrastructure systems that can be studied with DAD problem modeling include, but are not limited to, a road and highway network with a flow of traffic (Leblanc, 1975) or an energy distribution system with flows of gas or electricity (Brown et al., 2005).

McMasters and Mustin (1970) devised one of the first algorithms to solve (1.2b) for the optimal attack or interdiction of a supply network. When applied to network flow

problems, the attacker model of (1.2b) is also historically referred to as the network interdiction problem (Golden, 1978). There are at least three predominant methods for solving an AD problem. The decomposition method of Benders' (1962) examines one set of variables in a sub-problem and the other set of variables in a master problem. Fulkerson and Harding (1977) showed how to apply duality theory to transform the attacker model into a single level of optimization. A third solution method is implicit enumeration (IE) of all attack strategies (Scaparra & Church, 2008). We utilize all of these approaches in our research.

The network interdiction problem is a computationally difficult problem. The computational complexity of (1.2b) has been proven to be in the class of NP-Hard problems (Hansen, Jaumard and Savard, 1992). Wood (1993) showed that the network interdiction problem is NP-Complete.

The problem of optimal allocation of defense resources against an attacker in a bi-level defender-attacker (DA) optimization model has been in existence since the 1960s. Pugh (1964) considered the optimal deployment of Intercontinental Ballistic Missile (ICBM) defenses against an adversary designing a nuclear attack. Pugh's (1964) fundamental insights of the goals of the defender and the attacker are still relevant today:

It usually must be assumed that a prospective attacker will utilize knowledge of the defense deployment in designing an attack. Hence the defense problem is not to minimize damage resulting from a specific attack, but to choose that allocation of defense resources that will give best protection when an attack is optimized against whatever defenses are chosen. (p. 543)

The shortcoming of the bi-level DA model for applications that we consider is that it only models actions of the defender (W) and the attacker (X). The subsequent reaction of the operator of the system (Y) after the defenses and attacks are not explicitly optimized in the bi-level DA model. The DA model is shown in (1.2c).

$$\min_{W \in \Psi} \max_{X \in \Xi} f(W, X) \tag{1.2c}$$

The order of the max and min operators in the problem formulation determines the structure of the problem, since the entity that wishes to maximize damage to the system is separate and distinct from the entity that wants to operate and/or defend the system. Danskin (1967) showed the placement of the operators in the bi-level optimization problem formulation is essential, since a max-min problem is not equivalent, in general, to a min-max problem. The standard result is in Equation (1.2d).

$$\max_{X \in \Xi} \min_{Y \in Y(\hat{W})} f(\hat{W}, X, Y) \leq \min_{Y \in Y(\hat{W})} \max_{X \in \Xi} f(\hat{W}, X, Y). \quad (1.2d)$$

An AD model is a sequential two-person zero-sum game (Washburn & Wood, 1995). In the worst case, an attacking enemy will have perfect knowledge of all defenses prior to commencing an attack. Implicit in this assumption is the game theoretic notion of perfect information, that both the attacker and the defender know all of the relevant information about the capabilities of themselves and their adversary (Owen, 1995). It is also assumed that both the attacker and defender are rational and will select the best strategies to optimize their objectives (Owen, 1995). It is possible that attackers may not always behave rationally and choose the optimal worst-case attack (Smith et al., 2007). However, the assumption of rationality is always conservative in the zero-sum case.

Early attempts at solving the DAD problem focused on reducing the tri-level model to a bi-level model. Wood (1993) showed the dual of the inner operator problem can be exploited when modeled with continuous variables. The linear programming dual of the inner operator problem is a maximization that coincides with the maximization of the attacker model. Thus, the tri-level DAD problem can be transformed to a bi-level problem. The DAD problem becomes a DA model in (1.2e) with operator model dual variables for nodes (π) and arcs (α).

$$\min_{W \in \Psi} \max_{X, \pi, \alpha \in \Xi} f(W, X, \pi, \alpha) \quad (1.2e)$$

Expression (1.2e) is a challenging bi-level mixed integer program, since both the minimization variables (W) and the maximization variables (X) are discrete. Morton, Pan

and Saeger highlight the difficulty of (1.2e) by stating, “The problem is formulated with a nested ‘min-max’ structure, which is not amenable to solution through standard optimization algorithms” (2007, p. 6). A shortcoming of reducing the DAD problem to a bi-level model is that such a transformation is only possible when the inner operator model of (1.2a) has all continuous variables. If the operator model does not contain continuous variables, then the bi-level reduction of the DAD problem of (1.2e) cannot be performed. Chapters II and III will take advantage of the simplification of the DAD model in (1.2e). We also consider a DAD model where the reformulation in (1.2e) cannot be performed in Chapter IV.

Tri-level optimization for the defense, interdiction and operation of systems have appeared in research as early as Israeli (1999). Israeli and Wood briefly mention the “system defense problem” of hardening systems against attack where “the network user interdicts the interdictor” with a nested decomposition scheme, but further details are not provided (2002, p. 111). As recently as ten years ago, real-world instances of tri-level DAD models were considered extremely difficult to solve (Brown, Carlyle, Salmeron and Wood, 2005). Scaparra and Church declared, “To the best of the authors’ knowledge, no efficient algorithms have been proposed to deal with three stage [defender-attacker-defender] problems” (2008, p. 1906). However, advances in computing power and modeling techniques allow larger tri-level DAD problems to be solved in a reasonable amount of time. The tri-level model incorporates everything from the bi-level model, along with the addition of a distinct action for the defender of the system in anticipation of an attack. The defender chooses a defense option subject to a set of constraints that either expand the system by constructing additional new edges or harden the system by reducing the impact of an enemy attack against an edge. The defender model of (1.1c) is restated as (1.2f), and is also known as the *DAD problem* (Alderson et al., 2011).

$$\min_{W \in \Psi} \max_{X \in \Xi} \min_{Y \in Y(W)} f(W, X, Y) \quad (1.2f)$$

Israeli observed that (1.2f) is “NP-Hard and not known to be in NP” (1999, p. 61). Expression (1.2f) incorporates the network interdiction problem of (1.2b) as a special

case when there are zero defenses. Evaluation of any feasible defense requires the solution of an NP-Hard interdiction problem to compute an objective function value.

Algorithms to find optimal solutions to the defender model of (1.2f) as a tri-level optimization model were conjectured by Israeli (1999), but not implemented. Successful implementations of tri-level DAD algorithms are a recent development (Alderson et al., 2011 & 2014). Instances of an original Soviet Union railway problem from Harris and Ross (1955) have been solved using tri-level optimization by Alderson, Brown, Carlyle and Cox (2013). Tri-level optimization has also been applied to a fuel distribution network (Alderson et al., 2015). We use the fuel distribution network from these authors paper to test and validate some of our research results in Chapters I–III.

C. DAD NESTED DECOMPOSITION ALGORITHMS

As a starting point, we implement decomposition algorithms based on Alderson et al. (2011 & 2014) of a particular tri-level DAD problem instance using the General Algebraic Modeling Software (GAMS) version 24.5 with the IBM CPLEX solver. The DAD decomposition algorithms we implemented are similar to those employed by Benders (1962) with one key difference. Benders (1962) decomposition uses continuous variables in the sub problem and discrete variables in the master problem. Our decomposition procedure features binary defense variables (W), binary attack variables (X) and integer flow variables (Y). The differences in variable types require modifications to traditional decomposition techniques. We present the formulation of a DAD minimum cost flow problem and our implementation of decomposition in this section as an example of DAD nested decomposition. The nested decomposition algorithms discussed here follow the structure of Alderson et al. (2011 & 2014).

1. DAD Minimum Cost Flow Model Formulation

Alderson et al. (2014 & 2015) present a DAD model for defending, attacking and operating a minimum cost network flow model. We formulate the DAD minimum cost flow problem in order to explain the inner workings of nested decomposition. We use this notation as a common lexicon in Chapters I–III. Chapter IV has minor changes to this notation, as a different DAD problem is considered.

Sets:

$n \in N$	Nodes of the network (Note: i and j are used as aliases for n).
$(i, j) \in E$	Undirected edges of the network.
$(i, j) \in A$	Directed arcs of the network.
$d \in D$	Defense options available for undirected edges (i, j) . (Note: d_0 represents option of no defenses.)
$k \in K$	Iterations of inner decomposition.
$k' \in K'$	Iterations of outer decomposition.

Data:

c_{ijd}	Cost to move one unit of flow on arc (i, j) under defense plan d .
u_{ijd}	Maximum capacity of undirected edge (i, j) under defense plan d .
q_{ijd}	Per unit penalty cost to use attacked arc (i, j) with defense plan d .
p_n	Per unit penalty cost for not satisfying demand at node n .
r_{ij}	Cost to attack undirected edge (i, j) .
h_{ijd}	Cost to protect or build undirected edge (i, j) under defense plan d .
$demand_n$	Resources at node n . (positive is supply, negative is demand)
$attack_budget$	Maximum number attacks the attacker can afford.
$defense_budget$	Maximum number of defenses the defender can afford.

Decision Variables (units):

S_n	Amount of unfulfilled demand (shortage) at node n . (Units: items, etc.)
$S_{nk'}$	Unfulfilled demand at node n during outer decomposition iteration k' .
W_{ijd}	Binary; 1 if edge (i, j) chooses defense option d , 0 otherwise. (Unit less)
X_{ij}	Binary; 1 if edge (i, j) is attacked, 0 otherwise. (Unit less)
Y_{ijd}	Amount of flow on arc (i, j) with defense d . (units: barrels, items, etc.)
$Y_{ijk'}$	Flow on arc (i, j) with defense d during outer decomposition iteration k' .
π_n	Dual variable associated with each network node n .
α_{ijd}	Dual variable associated with each network arc (i, j) under defense plan d .
Z_{AD}	Objective function value of inner decomposition
Z_{DAD}	Outer decomposition objective function value

Additional Notation:

\hat{W}_{ijd}	Fixed value 1 if arc (i, j) has defense option d assigned; 0 otherwise.
-----------------	---

\hat{X}_{ij}	Fixed value 1 if edge (i, j) is assigned to be attacked; 0 otherwise.
$\hat{X}_{ijk'}$	Fixed value 1 if edge (i, j) is attacked in outer iteration k'; 0 otherwise.
\hat{Y}_{ijd}	Fixed amount of flow assigned to arc (i, j) with defense d in iteration k.
\hat{S}_{nk}	Fixed amount of unfulfilled demand assigned to node n in iteration k.

Formulation:

$$\min_{W_{ijd}} \max_{X_{ij}} \min_{S_n, Y_{ijd}} \sum_{d \in D} \sum_{(i,j) \in E} \left(c_{ijd} + (q_{ijd} \cdot X_{ij}) \right) \cdot Y_{ijd} + \left(c_{jid} + (q_{jid} \cdot X_{ij}) \right) \cdot Y_{jid} + \sum_{n \in N} S_n \cdot p_n, \quad (1.3a)$$

Subject To:

$$\sum_{d \in D} \sum_{(n,j) \in A} Y_{njd} - \sum_{d \in D} \sum_{(i,n) \in A} Y_{ind} - S_n \leq demand_n \quad \forall n \in N, \quad (1.3b)$$

$$Y_{ijd} + Y_{jid} \leq u_{ijd} \cdot W_{ijd} \quad \forall (i, j) \in E, d \in D, \quad (1.3c)$$

$$\sum_{(i,j) \in E} r_{ij} \cdot X_{ij} \leq attack_budget, \quad (1.3d)$$

$$\sum_{d \in D: (i,j) \in E} \sum_{d \neq d_0} h_{ijd} \cdot W_{ijd} \leq defense_budget, \quad (1.3e)$$

$$\sum_{d \in D} W_{ijd} = 1 \quad \forall (i, j) \in E, \quad (1.3f)$$

$$Y_{ijd} \geq 0 \quad \forall (i, j) \in A, d \in D, \quad (1.3g)$$

$$S_n \geq 0 \quad \forall n \in N, \quad (1.3h)$$

$$X_{ij} \in \{0,1\} \quad \forall (i, j) \in E, \quad (1.3i)$$

$$W_{ijd} \in \{0,1\} \quad \forall (i, j) \in E, d \in D. \quad (1.3j)$$

The objective function (1.3a) expresses the three levels of optimization. The innermost minimization operator seeks to minimize the cost of flows (Y) and shortages (S) on the network after defenses and attacks are fixed. The middle maximization operator has the objective of maximizing the cost of the resulting optimal flow, by attacking arcs for a given defense plan. The outer minimization operator chooses the optimal defense (W) to minimize the worst-case cost over all attacks. Equation (1.3b) represent balance of flow equations for each node in the network, with an elastic variable compensating for unfulfilled demand. Equations (1.3c) are bi-directional (i.e., shared) capacity constraints for each edge in the network, forcing total flow on an edge to be less than the maximum capacity (u) for the edge under a particular defense option (d). Equation (1.3d) is a budget constraint on the attacker to ensure that the attacker can only

attack as many edges as he or she can afford. Equation (1.3e) is a budget constraint on the defender which ensures the amount of network expansion or hardening stays within a budget. Equation (1.3f) ensures each edge of the network can only have one defense option chosen. Equations (1.3g) – (1.3j) enforce variable types.

At a minimum, we have two possible defense choices for each arc. Defense choices in our application either leave an arc undefended (d_0) or choose to defend an arc (d_1). Our development of DAD CSP is based on two defense choices. However, our formulation can support more than two defense choices.

We can now define the constraint sets from the previous section in the context of the DAD minimum cost flow problem. The operator model constraint set (Υ (W)) consists of Equations (1.3b), (1.3c), (1.3g), and (1.3h). The attacker model constraint set (Ξ) consists of Equations (1.3d) and (1.3i). The defender model constraint set (Ψ) consists of Equations (1.3e), (1.3f), and (1.3j). Although Equation (1.3c) contains defense variables (W), we consider it to define the feasible region of the path variables (Y), and not to place any restrictions on the defense variables (W) themselves. Since the defender chooses the defense variables (W) before the path variables (Y) are determined, the operator model is feasible for any feasible values of the defense variables (W) in the defender model.

2. AD Inner Decomposition

The structure of the inner decomposition is a bi-level maxi-min AD optimization problem. The inner decomposition fixes the values of the defense variables (W), resulting in an AD inner decomposition problem in the form of (1.2b).

AD Inner Decomposition Problem Formulation:

$$\max_{X_{ij}} \min_{S_n, Y_{jd}} \sum_{d \in D} \sum_{(i,j) \in A} \left(c_{ijd} + \left(q_{ijd} \cdot \left(X_{ij} \Big|_{i < j} + X_{ji} \Big|_{i > j} \right) \right) \right) \cdot Y_{ijd} + \sum_{n \in N} S_n \cdot p_n, \quad (1.4a)$$

Subject to:

$$\sum_{d \in D} \sum_{(n,j) \in A} Y_{njd} - \sum_{d \in D} \sum_{(i,n) \in A} Y_{ind} - S_n \leq \text{demand}_n \quad \forall n \in N, \quad (1.4b)$$

$$Y_{ijd} + Y_{jid} \leq u_{ijd} \cdot \hat{W}_{ijd} \quad \forall (i,j) \in E, d \in D, \quad (1.4c)$$

$$\sum_{(i,j) \in E} r_{ij} \cdot X_{ij} \leq \text{attack_budget}, \quad (1.4d)$$

$$Y_{ijd} \geq 0 \quad \forall (i,j) \in A, d \in D, \quad (1.4e)$$

$$S_n \geq 0 \quad \forall n \in N, \quad (1.4f)$$

$$X_{ij} \in \{0,1\} \quad \forall (i,j) \in E. \quad (1.4g)$$

Expression (1.4a) maximizes the minimum cost of flow over the network. Equation (1.4b) is identical to the balance of flow Equations (1.3b). Equation (1.4c) is similar to Equation (1.3c) except the defense option is fixed in the inner decomposition. Equations (4d–4g) are carried over from the previous formulation.

The inner AD sub-problem has a maxi-min structure that can be solved with two different approaches. The first approach is a decomposition of the primal problem. The second approach takes advantage of continuous flow variables (Y) to formulate a dual integer linear program (ILP) with a single level of optimization. The dual problem has a maxi-max structure which replaces the inner stage of decomposition. Either approach to the inner AD problem will function as a sub-problem for the outer decomposition.

Solution elimination constraints (SEC) appear in both of the following approaches because it is possible that the inner decomposition can produce an attack that has been attempted previously (Alderson et al., 2014). If the repeated attack solution is utilized again in the outer decomposition, there is a possibility that the outer decomposition could cycle endlessly between previously found defense solutions. The SEC function as anti-cycling constraints for the outer decomposition. SEC are only activated in a decomposition iteration if the attack found in the current iteration was also found in a previous iteration. Otherwise, the SEC is not included in an iteration. Activation of the SEC in an outer decomposition iteration forces the outer decomposition master problem to produce a new and different solution that breaks the cycling of the outer decomposition (Alderson et al., 2011).

a. AD Primal Decomposition Formulation

The maxi-min linear decomposition of the AD inner problem can be modeled by introducing new decision variables as described in Rardin (1998, pp. 159–160). The AD

primal inner subproblem represents the operator model, where the defender seeks to minimize the cost of moving resources from sources to destinations in a network with fixed defenses (W) and attacks (X).

AD Minimum Cost Flow Primal Inner Subproblem Formulation:

$$\min_{S_n, Y_{ijd}} \sum_{(i,j) \in A} \left(c_{ijd} + \left(q_{ijd} \cdot \left(\hat{X}_{ij} \Big|_{i < j} + \hat{X}_{ji} \Big|_{i > j} \right) \right) \right) \cdot Y_{ijd} + \sum_{n \in N} S_n \cdot p_n, \quad (1.5a)$$

$$\sum_{d \in D} \sum_{(n,j) \in A} Y_{njd} - \sum_{d \in D} \sum_{(i,n) \in A} Y_{ind} - S_n \leq demand_n \quad \forall n \in N, \quad (1.5b)$$

$$Y_{ijd} + Y_{jid} \leq u_{ijd} \cdot \hat{W}_{ijd} \quad \forall (i,j) \in E, d \in D, \quad (1.5c)$$

$$Y_{ijd} \geq 0 \quad \forall (i,j) \in A, d \in D, \quad (1.5d)$$

$$S_n \geq 0 \quad \forall n \in N. \quad (1.5e)$$

Expression (1.5a) is the operator objective, which is to minimize the cost of moving resources from source to destination in the network after defenses and attacks have occurred. Equation (1.5b)-(1.5e) restate their respective constraints from system of Equations (1.4). The set K represents iterations over an inner AD model. The result of each iteration of the AD primal inner sub problem is a lower bound on the inner decomposition. Each time the objective function is better than the existing lower bound, the bound is updated.

The AD primal inner master problem represents the *attacker* problem, where the attacker has a fixed budget to damage or degrade the network in order to maximize the cost of moving resources from sources to destinations. In the AD primal inner master problem, the values of the flows (Y) and shortages (S) are fixed values input from the primal inner sub problem.

AD Minimum Cost Flow Primal Inner Master Problem Formulation:

$$\max_{X_{ij}, Z_{AD}} Z_{AD}, \quad (1.5f)$$

$$Z_{AD} \leq \sum_{d \in D} \sum_{(i,j) \in E} (c_{ijd} + q_{ijd} \cdot X_{ij}) \cdot \hat{Y}_{ijdk} + (c_{jid} + q_{jid} \cdot X_{ij}) \cdot \hat{Y}_{jids} + \sum_{n \in N} \hat{S}_{nk} \cdot p_n \quad \forall k \in K, \quad (1.5g)$$

$$\sum_{(i,j) \in E} r_{ij} \cdot X_{ij} \leq attack_budget, \quad (1.5h)$$

$$X_{ij} \in \{0,1\} \quad \forall (i,j) \in E, \quad (1.5i)$$

$$\sum_{\substack{(i,j) \in E: \\ \hat{X}_{ijk}=0}} X_{ij} + \sum_{\substack{(i,j) \in E: \\ \hat{X}_{ijk}=1}} (1 - X_{ij}) \geq 1 \quad \forall k' \in K. \quad (1.5j)$$

Expressions (1.5f) and (1.5g) work together to create the objective of maximizing the cost of flow through the network through the choice of attacks. Equation (1.5g) represents the restriction of the problem through “cuts” of the feasible region performed by the choice of attack variables (X) in each iteration (k) of inner decomposition. Equations (1.5h) and (1.5i) restate Equations (1.3d) and (1.3i).

The AD master problem produces an upper bound on the inner decomposition. During iterations that improve the upper bound, the attack associated with that iteration is saved as the best known attack (X*). When the inner decomposition completes because either the inner lower and upper bounds converge or the number of iterations (k) has been reached, then the best known attack (X*) is given as an input to the DAD outer decomposition master problem.

Equation (1.5j) is the SEC for the outer decomposition. When the inner decomposition produces a best attack (X*) that is the same as a best attack from a previous outer iteration (k'), then there is a potential for cycling of the outer decomposition. In order to prevent cycling, the SEC of Equation (1.5j) is temporarily activated in the inner decomposition to ensure that an alternate best attack (X*) is produced that has never been produced in a previous outer iteration (k').

b. AD Dual ILP Formulation

In order to eliminate inner iterations, we can alternatively formulate the AD dual ILP problem by taking the dual of the innermost minimization sub problem. The dual problem exists only when the network structure of the problem allows the flow variables (Y) to be treated as continuous variables. From the primal formulation, we note that Equation (1.3b) has dual variable $[-\pi_n]$ which represents each node and Equation (1.3c) has dual variable $[-\alpha_{ijd}]$ which represents each edge. We introduce both the dual variable for the node equations and the edge equations as negative values for convenience.

AD Minimum Cost Flow Dual ILP Formulation:

$$\max_{X_{ij}, \alpha_{ijd}, \pi_n} - \sum_{n \in N} demand_n \cdot \pi_n - \sum_{d \in D} \sum_{(i,j) \in E} u_{ijd} \cdot \alpha_{ijd} \cdot \hat{W}_{ijd}, \quad (1.6a)$$

Subject to:

$$-\pi_i + \pi_j - \alpha_{ijd} \Big|_{i < j} - \alpha_{jid} \Big|_{i > j} \leq c_{ijd} + \left(q_{ijd} \cdot \left(X_{ij} \Big|_{i < j} + X_{ji} \Big|_{i > j} \right) \right) \quad \forall (i, j) \in A, d \in D, \quad (1.6b)$$

$$\pi_n \leq p_n \quad \forall n \in N, \quad (1.6c)$$

$$\sum_{(i,j) \in E} r_{ij} \cdot X_{ij} \leq attack_budget, \quad (1.6d)$$

$$-\alpha_{ijd} \geq 0 \quad \forall (i, j) \in E, d \in D, \quad (1.6e)$$

$$-\pi_n \geq 0 \quad \forall n \in N, \quad (1.6f)$$

$$X_{ij} \in \{0, 1\} \quad \forall (i, j) \in E, \quad (1.6g)$$

$$\sum_{\substack{(i,j) \in E: \\ \hat{X}_{ijk}=0}} X_{ij} + \sum_{\substack{(i,j) \in E: \\ \hat{X}_{ijk}=1}} (1 - X_{ij}) \geq 1 \quad \forall k' \in K. \quad (1.6h)$$

Expression (1.6a) is the dual objective function, which seeks to maximize the cost flow for an attack based on a given defense strategy. Equation (1.6b) is the dual for each node, which occurs over the set of all arcs and defense strategies. Equation (1.6b) has variables with set restrictions because of the relationship between the decision variables and the undirected edge set (E) vs. the directed arc set (A). Note that the node dual variables (π_n) exist for all nodes, and the edge dual variables (α_{ijd}) exist over the set of all edges (E) and defense options (D). However, the edge dual variables do not exist over the set of all directed arcs A. Equation (1.6c) is associated with shortages at each node, which effectively places an upper limit on the node dual variables. Equation (1.6d) restates Equation (1.3d). The dual decision variables have non-negativity restrictions in Equations (1.6e) and (1.6f) because the dual variables are introduced with negative values. Equation (1.6g) ensures attack variables are binary values. Equation (1.6h) is a SEC that is activated as necessary to prevent cycling in the outer master problem as described in the explanation of Equation (1.5j). The output of the AD dual ILP is an attack (X) that is input to the outer decomposition.

3. DAD Outer Decomposition

The outer decomposition master problem takes the best known attack (X*) from either version of the inner AD problem as input to the DAD outer master problem. Each

outer decomposition iteration (k') determines the best choice of defenses (W) in response to the attacks (X) that it has observed so far. Furthermore, each outer decomposition iteration (k') must also determine the best combination of flow variables (Y) in conjunction with the defense variables (W).

DAD Minimum Cost Flow Outer Master Problem Formulation:

$$\min_{W_{ijd}, Y_{ijk'}, S_{nk'}} Z_{DAD}, \quad (1.7a)$$

Subject to:

$$Z_{DAD} \geq \sum_{d \in D} \sum_{(i,j) \in E} (c_{ijd} + q_{ijd} \cdot \hat{X}_{ijk'}) Y_{ijk'} + (c_{jid} + q_{jid} \cdot \hat{X}_{ijk'}) Y_{jkd'} + \sum_{n \in N} S_{nk'} \cdot p_n \quad \forall k' \in K, \quad (1.7b)$$

$$\sum_{d \in D} \sum_{(n,j) \in A} Y_{njdk'} - \sum_{d \in D} \sum_{(i,n) \in A} Y_{indk'} - S_{nk'} \leq demand_n \quad \forall n \in N, k' \in K, \quad (1.7c)$$

$$Y_{ijk'} + Y_{jkd'} \leq u_{ijd} \cdot W_{ijd} \quad \forall (i,j) \in E, d \in D, k' \in K, \quad (1.7d)$$

$$S_{nk'} \geq 0 \quad \forall n \in N, k' \in K, \quad (1.7e)$$

$$Y_{ijk'} \geq 0 \quad \forall (i,j) \in A, d \in D, k' \in K, \quad (1.7f)$$

$$W_{ijd} \in \{0,1\} \quad \forall (i,j) \in E, d \in D. \quad (1.7g)$$

Expressions (1.7a) and (1.7b) work together to create the objective of minimizing the cost of flow through the network based on fixed values of attacks (X) obtained from the inner AD problem. Equation (1.7b) represents the restriction of the outer DAD problem through *cuts* of the feasible region performed by the choice of defense (W), flow (Y) and shortage (S) variables in each iteration (k') of outer decomposition. Equation (1.7c) expands Equation (1.3b) to include flow (Y) and shortage (S) variables for each round of outer decomposition (k'). Equation (1.7d) ensures the flow (Y) variables do not exceed their maximum capacity for an edge for every iteration of outer decomposition (k'). Equations (1.7e)—(1.7g) enforce variable types.

The output of the DAD outer master problem is a set of defense variables and an objective function value. The outer master problem updates the outer decomposition lower bound when the objective function value is greater than all previous outer iteration values. Whenever the outer lower bound is updated, the defense associated with that outer iteration (k') is retained as the best known defense (W^*). When the outer upper and

60 minutes. We can solve the same 36 instances using the dual ILP and decomposition in roughly 8 minutes. For this test case, the difference in solution speed is dramatic, with the dual ILP and decomposition method clearly superior.

For the 36 problem instances, both implementations of the DAD decomposition produced the same objective function values as Alderson et al. (2015). In some cases, more than one combination of defenses and/or attacks can produce the same optimal solution value. Table 1 shows a portion of the results for the DAD decomposition applied to the network of Figure 2 for varying attack strategies when the defense budget is fixed at four units. Table 2 takes the opposite view of supposing that the defender has intelligence that the attacker budget is fixed at four units, and there is a varying defense budget available. Edges not appearing in Tables 1 and 2 have no recommendations for attack or defense. Green colored cells indicate defensive action and red colored cells indicate attack action in Tables 1 and 2.

Table 1. Test Network Solution with Fixed Defense Budget of Four Units

Defense Budget= 4	Attack Budget					
Edge	0	1	2	3	4	5
[3,8]	Build	Build	Build	Build	None	None
[5,10]	Build	Build	Build	None	Build	None
[10,11]	None	None	None	Protect	Protect	Protect
[10,13]	None	Attack	None	Protect	Attack	Protect
[1,5]	None	None	None	None	Protect	None
[2,7]	None	None	None	None	None	Protect
[7,8]	None	None	None	None	Attack	Protect
[8,12]	None	None	Attack	Attack	None	Attack
[11,12]	None	None	Attack	None	None	Attack
[1,2]	None	None	None	Attack	None	Attack
[9,13]	None	None	None	Attack	None	Attack
[11,15]	None	None	None	None	Attack	Attack
[5,9]	None	None	None	None	Attack	None
Flow Cost	20	23	37	47	58	65

Table 2. Test Network Solution with Fixed Attack Budget of Four Units

Attack Budget= 4	Defense Budget					
Edge	0	1	2	3	4	5
[3,8]	None	None	None	None	None	Build
[5,10]	None	None	None	Build	Build	None
[10,11]	Attack	None	Attack	Protect	Protect	None
[10,13]	Attack	Protect	Attack	Attack	Attack	Protect
[1,5]	None	None	None	Attack	Protect	None
[2,7]	Attack	None	None	None	None	None
[7,8]	None	Attack	None	Attack	Attack	None
[8,12]	Attack	Attack	Attack	None	None	Protect
[11,12]	None	None	None	None	None	None
[1,2]	None	None	None	None	None	Attack
[9,13]	None	Attack	Attack	None	None	Protect
[11,15]	None	None	None	None	Attack	Attack
[5,9]	None	None	None	None	Attack	Attack
[13,14]	None	None	None	None	None	Attack
[9,10]	None	None	Build	None	None	None
[6,10]	None	Attack	None	Attack	None	None
Flow Cost	113	82	74	66	58	53

E. SUMMARY OF CONTRIBUTIONS

The existing literature defines the basics of the DAD problem applied to infrastructure systems, but there are many unexplored topics associated with this problem. Literature provides formulations only for minimum cost flows and shortest paths. Literature also provides a nested decomposition approach with SEC to obtain an optimal solution to a DAD problem instance. This dissertation seeks to expand knowledge of DAD problems beyond the one solution approach. We address three interesting areas of the DAD problem that have not been developed in existing literature.

1. Efficient Enumeration Algorithms for the DAD Model

Chapter II explores the ability to generate multiple optimal solutions and near optimal solutions to a DAD minimum cost flow problem instance. The knowledge that equivalent optimal solutions exist can be useful information to a decision maker. For example, if an analyst has a defense budget of three units to defend a system against a worst-case attack, it could be helpful to know if there is only one possible optimal

defense or more than one equivalent optimal defenses. Many equivalent optimal solutions can present a range of equally good choices to defend a system. Similarly, a decision maker may want to know all of the defense options that are within a small fixed percentage of the optimal solution. Decision makers could then use a range of options to help weigh other factors to the system that cannot be modeled, but are still important to consider. In the real world, some defenses that appear to be equal on paper may have other considerations that make one set of defenses preferable to another set of defenses, even though they both produce the same objective function value. Traditionally, finding many optimal and near optimal solutions would require resolving the DAD problem for each additional solution that is desired. We design an algorithm that can find all of the optimal and near optimal solutions to a DAD minimum cost flow problem efficiently.

Chapter II harnesses the theory of implicit enumeration (IE) to efficiently generate all equivalent optimal solutions. IE can also find all near optimal solutions for any specified solution tolerance. Implicit enumeration works by restricting and subsequently relaxing the defense budget of the original problem instance. IE reduces the tri-level DAD problem into a group of simpler bi-level AD sub problems arranged in a tree structure in order to systematically build solutions. IE avoids the possibility of listing every possible solution by rejecting combinations of defenses and attacks that could not lead to an optimal or near optimal solution. The original application of IE to DAD optimization problems was developed by Scaparra and Church (2008). These authors present the use of IE in the context of defending and attacking the placement of warehouse facilities. Our contribution is to adapt, generalize and expand on known IE theory and apply it to the tri-level DAD minimum cost flow problem.

2. Impact of Nested Defenses on Optimality

Chapter III investigates system defense planning under uncertain budget scenarios, and how that uncertainty impacts optimality. It may be possible that a system analyst may not be certain about the budget for his or her defenses, the attack budget of the adversary, or both. When faced with budget uncertainty, the analyst may produce a list of prioritized defenses for the system. The prioritized list would include the best

defense, second best defense, third best defense, and so on. A prioritized list becomes a set of *nested defenses* to be chosen when the budget scenario becomes known. We use the term *nested* to refer to a monotonic sequence of sets, where each set of defenses for a particular budget scenario contains the set of defenses for the next smaller budget scenario. The use of nested defenses is attractive to decision makers because it simplifies the complex analysis of choosing the best defenses under uncertain budgetary scenarios into a single list.

The set of nested defenses almost always is not the same as the set of optimal defenses for any particular defense and attack budget. It is clear in existing literature that the best defenses chosen for a defense budget of two units is usually not a subset of the defenses chosen for a defense budget of three units, and so on. An example is shown in Figure 2 from Alderson et al. (2015). Figure 2 shows that if the defense budget is known, but the attack budget is unknown for the fuel network, then the elements of an optimal defense change. In Figure 2, the panels labeled (b) to (e) show that even when the defense budget is unchanged at four units, each change in the attack budget from two to five units results in a different set of optimal defense elements chosen and a different set of worst-case attacks. The optimal defenses for the network are shown as blue edge lines, and the worst-case attacks are shown as small explosions on edges. A similar effect exists when the defense budget is unknown and the attack budget is known.

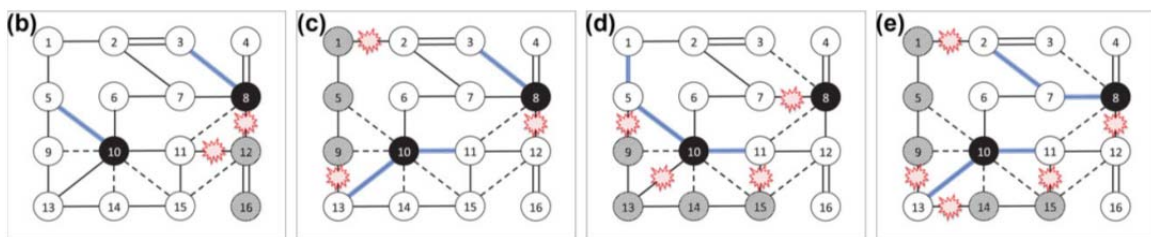


Figure 2. Optimal Defenses Change as the Attack Budget Changes. Source: Alderson et al. (2015).

Chapter III uses the same network as Alderson et al. (2015) to investigate the effects of nested defenses. We examine how nested defenses differs from the optimal solution for any particular defense and attack budget scenario and, in particular, we

quantify the “cost” of requiring a set of defense plans to be monotonic. A gap exists in current literature for how to address the situation when both defense and attack budgets are uncertain, and we propose a strategy for that situation. We examine various heuristic nesting strategies and compare their performance against the known optimal solutions. In order to find the best possible nested defense, we develop a new parametric programming formulation of the DAD minimum cost flow problem that accepts uncertainty in the defense and attack budgets. This new formulation generates a nested defense that is as close as possible to the set of optimal non-nested solutions.

3. Incorporating Heuristics to Speed Up DAD Solution Algorithms

Chapter IV develops a DAD model for the constrained shortest path (CSP) problem. To the best of our knowledge, solution procedures for the DAD CSP problem do not exist in the literature. The challenge associated with the CSP problem is that the side constraint placed on a shortest path problem breaks the network structure of the problem. Loss of the network structure of the problem results in a much more difficult binary program because the path variables (Y) cannot be treated as continuous. First, we formulate and solve instances of the DAD CSP by adapting existing DAD nested decomposition techniques from the literature. We find the point where existing DAD solution techniques become either intractable or excessively time intensive to solve the problem. Our goal is to develop alternatives and improvements to the nested decomposition approach to solve DAD CSP problem instances more efficiently.

We improve the time it takes to find a solution to a DAD CSP instance by applying Lagrangian relaxation. Ahuja, Magnanti and Orlin (1993) explain the use of Lagrangian relaxation on a regular CSP problem instance involving a small test network. Fisher (1981) notes that the Lagrangian relaxation is able to find a greatest lower bound on an original problem instance. Since only a greatest lower bound can be determined, we observe Lagrangian relaxation is a heuristic approach to finding a solution.

We expand the Lagrangian relaxation techniques found in existing literature to include defending and attacking the CSP problem. We develop two different alternative algorithms for Lagrangian relaxation of DAD CSP. The first algorithm closes the gap

created by the Lagrangian lower bound by using path enumeration techniques proposed by Carlyle, Royset and Wood (2008). The second algorithm takes advantage of the fact that iterative Lagrangian relaxation searches for the greatest lower bound and the attacker model seeks to maximize the length of the shortest path. These similar goals allow us to combine the attacker problem and Lagrangian relaxation into a single mathematical model. Both algorithms are able to find a heuristic solution to DAD CSP quickly.

We also develop algorithms to combine our heuristic approaches with traditional methods to obtain provably optimal or near optimal solutions. By themselves, our improvements to Lagrangian relaxation remain a heuristic approach to the DAD CSP problem. The goal of our combined algorithms is to reduce the time required to obtain an optimal or approximately optimal solution to a DAD CSP instance when compared to regular nested decomposition. We test our algorithms on medium and large networks to show the scalability of our innovative approaches to solve the DAD CSP problem. Our results show that our new algorithms are significantly faster than regular nested decomposition to find an optimal or near optimal solution to a DAD CSP instance.

II. IMPLICIT ENUMERATION OF DAD PROBLEM

A. IMPLICIT ENUMERATION THEORY

Wagner (1975) gives a general goal for any enumeration algorithm as follows: “What we want are techniques that partially enumerate a manageable number of possibilities and implicitly enumerate the rest” (p. 473). The method of systematically discarding provably suboptimal combinations of decision variables is called Implicit Enumeration (IE). Formally, IE refers to “a systematic evaluation of all possible [feasible] solutions without explicitly evaluating all of them” (INFORMS, 2015). We develop an IE algorithm for defender-attacker-defender (DAD) problems that enumerates defense plans in a specific order. We use a bounding argument based on evaluating the resulting attacker-defender (AD) subproblem to significantly reduce the amount of enumeration required.

IE represents an alternative algorithm to obtain solutions for DAD problem instances. IE does not use the traditional nested decomposition algorithm in Chapter 1. Our enumeration algorithm manipulates the defense budget to restrict and relax the feasible region of the defense variables (Ψ) in DAD model (2.1a). Our IE algorithm systematically fixes defense variables (W) and solves a series of bi-level AD models shown in (2.1b). The IE algorithm can also find multiple equivalent solutions to a problem instance when they occur. Similarly, if all solutions within a given percentage of the optimal solution are desired, IE can generate all of possible objective function values and the combinations of variables needed for each solution.

$$\min_{W \in \Psi} \max_{X \in \Xi} \min_{Y \in Y(W)} f(W, X, Y) \quad (2.1a)$$

$$\max_{X \in \Xi} \min_{Y \in Y(\hat{W})} f(\hat{W}, X, Y) \quad (2.1b)$$

The IE algorithm creates a tree of AD subproblems beginning with the restriction of the defense budget to zero units at the root node, followed by incremental relaxations of the defense budget at subsequent nodes. Every node in our enumeration tree represents

an instance of an AD subproblem with a fixed defense plan (\hat{W}) that produces a corresponding worst-case attack (X). The children of this parent node are specified by adding one more fixed defense (\hat{W}) to the defense plan if its corresponding AD subproblem results in either a change to the worst-case attack (X) or an improvement of the objective function value. Enumeration of fixed defenses in child nodes is ended when the relaxed defense budget reaches its original value.

In this chapter, we utilize the DAD minimum cost flow problem formulation and the 16 node fuel transportation network (Alderson et al., 2015) described in Chapter I. In the remainder of this chapter, we assume that there are exactly two defense options for each edge: d_0 means *no defense* (or *no construction* for an edge that does not have capacity in the undefended case) and d_1 represents the sole defense option for any edge that can be defended or built. We use the term *defended* to refer to edges for which we choose option d_1 and “undefended” for edges which we choose option d_0 . In cases where we want to model more than these two defense options for an edge, our discussion generalizes in a straightforward way.

1. Enumeration of Defenses in the Literature

Scaparra and Church (2008) develop an enumeration algorithm to solve a specific optimization problem called the “R-Interdiction Median with Fortification” (RIMF) problem, which is concerned with the defense of a limited number of facilities against attack. These authors uses duality within (2.1b) to treat a tri-level optimization problem with a bi-level model in (2.1c).

$$\max_{X, \pi, \alpha \in \Xi} f(\hat{W}, X, \pi, \alpha) \quad (2.1c)$$

Scaparra and Church state “In our specific case, the attacker–user problem can be modeled as a single-level mixed-integer [program] (MIP), so that what is in principle a tri-level defender–attacker–user problem can be reduced to a bi-level min–max problem” (2008, p. 1906). Scaparra and Church take advantage of the operator model continuous variables (Y) in order to reduce the bi-level AD model (2.1b) into a single level

optimization model similar to (2.1c). IE has not been successfully implemented on a tri-level DAD optimization problem without using duality. Scaparra and Church state “To the best of the authors’ knowledge, no efficient algorithms have been proposed to deal with three-stage defender-attacker-user problems” (2008, p. 1906).

In the RIMF problem, Scaparra and Church (2008) observe that the optimal defense against a worst-case attack must include at least one of the items that would be attacked if the system was undefended. To prove their point, the authors choose a starting point in IE of an undefended system, where all of the defense variables are fixed to zero, and observe the worst-case attack. Next, the authors observe that the only way to prevent the worst-case attack from occurring again is to defend one of the items that was attacked in the undefended scenario. The authors reason that “this observation can be easily explained by noticing that if none of the facilities in the optimal interdiction set is protected, then it is still possible to interdict all of them and the worst possible case of interdiction is not prevented” (2008, p. 1910).

This observation serves as one of the key concepts for our IE theory. At each node of IE, a subproblem is formed and a worst-case attack is observed. The AD subproblem in RIMF is similar to (2.1c). In the RIMF problem, the next layer of the IE tree is created by enumerating a fixed defense (\hat{W}) for each of the observed attacks in an AD subproblem. The authors realize that any other defense that is not responding to a specific attack is not going to prevent that same attack from occurring again. Therefore, if the next defense chosen in the subsequent round of IE does not specifically prevent the worst attack from occurring again, then the selection of that particular defense at that particular time must be a suboptimal choice.

2. Contributions to IE Theory

To the best of our knowledge, this is the first IE scheme of its type designed for the general DAD model. We provide an enumeration scheme similar to that proposed by Scaparra and Church (2008), but enhanced to handle the more general case. Our research makes two advances in IE theory. First, we expand IE theory to include more general tri-level DAD models. Our algorithm has the ability to use the dual MIP approach (2.1c) and

it can also incorporate solution procedures using (2.1b) for other tri-level DAD models where the dual MIP technique cannot be used. In this case, IE of fixed defenses results in bi-level AD sub problems that can be solved with traditional methods.

Our second contribution to IE theory is that our algorithms include the ability for IE to include network design as a potential defense for a tri-level DAD model. We introduce the ability to add new edges as system defenses. In the RIMF problem, Scaparra and Church (2008) note that defending each of the worst-case attacks one by one will lead to changes in the worst-case attack, and therefore they should be the only candidates for defenses. In the more general DAD setting, however, a defense variable can refer to the creation of new components. If the capacity of an undefended arc is zero, but the capacity of a defended arc is non-zero, then that arc only exists if we defend it. We define new edge transport capacity (u) to be zero when the edge is not added to the network (d_0), and greater than zero when it is added to the network as a defense (d_1). In this case, our enumeration must allow that new edges can alter the performance of the system, and therefore can reduce the impact of the worst-case attack.

Proposition 2–1: The addition of new edges is a feasible defense strategy against worst-case attack.

In the RIMF problem, prevention of a worst-case attack must include defending at least one of the items in the worst-case attack. We prove that this observation does not hold for a general DAD problem instance where new edges can be used as defenses with a counterexample. We consider the test network with a defense budget of four units and attack budget of two units. The undefended attack plan would be to attack edges (2, 7) and (9, 13) resulting in a system cost of 62. The optimal defense is to build the new edges (3, 8) and (5, 10), which does not involve defending one of the original attacked edges. Furthermore, the optimal system cost has improved to 37 and the attackers new best attack plan is to interdict edges (8, 12) and (11, 12). The resulting best defense of the network no longer involves the original attacked edges, or any existing edge in the original network. The two graphs in Figure 3 show the results, with defended edges in blue and attacked edges with a red “X.”

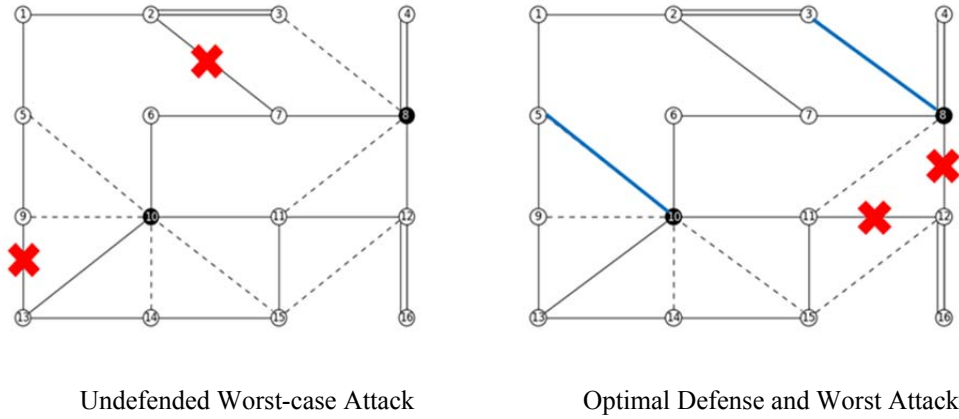


Figure 3. Optimal Defense with Addition of Edges on Test Network

Our counterexample shows there are other choices in the general DAD problem besides defending existing edges to prevent an observed worst-case attack. The challenge is to determine an efficient algorithm to enumerate defenses when new edges can be added to defend a system.

B. ENUMERATION TREE CONCEPTS

Our enumeration algorithm explores a tree of subproblems. Each node in the tree represents an AD subproblem with a specific fixed defense that is based on a restriction of the original defense budget. The AD subproblem instance for a node is a simplified bi-level model that is easier to solve than the original tri-level DAD optimization problem. Child nodes represent the inclusion of one additional fixed defense in response to the worst-case attack observed in a parent node AD subproblem.

The root node in our enumeration tree represents the initial restriction of the original DAD problem instance to a defense budget of zero units. Thus, the AD subproblem associated with the root node has no fixed defenses.

Branches from a parent node represent the choice of one new defense in response to the worst-case attack of an AD subproblem for a parent node. Any defense chosen as a branch must improve system performance or change the worst-case attack observed at the parent node. Thus, each branch represents one defense that is an appropriate response to a

worst-case attack of a parent node. Conversely, the branches that form a path from the root node to a particular node give the fixed defenses of the AD subproblem at that node.

Other nodes in the IE tree represent AD sub problems that relax the defense budget restriction at the root node. Any node can be identified by its depth and breadth. Depth is defined as the axial distance on the tree from the root node. The depth of the node in the tree tells the number of defenses that are fixed in its accompanying AD subproblem instance. Depth starts at zero at the root node because the root node has zero fixed defenses. Tree depth grows by one for each additional defense. For example, if a node has a depth of two in the IE tree, then the AD subproblem associated with that node has two fixed defenses. Breadth is defined as the number of nodes at a particular depth.

Child nodes on a tree are created when the remaining defense budget of a parent node is greater than the cost of adding an additional defense. Branches based on defenses against the worst-case attack of a parent node represent fixed defenses to be embodied in new AD sub problems associated with new child nodes. A leaf node on the tree occurs when the number of fixed defenses at a node exhausts the original defense budget. A defense budget could be exhausted at different depth levels for different nodes because the cost of defending existing edges or adding new edges may not be the same.

Traditionally, elimination of sections of an enumeration tree that cannot lead to an optimal solution is called pruning (Wolsey, 1998, p. 94). We seek to prune the IE tree by developing characterizations of a suboptimal solution to the DAD problem or corresponding AD sub problem. The pruned sections of the tree will not need to be explored because an optimal solution could not be developed from the nodes representing these AD subproblem instances. In this chapter, we develop some pruning rules that use the development of a decreasing upper bound on the DAD master problem to eliminate many possible defense combinations from consideration.

1. Data Structures at an IE Tree Node

To better understand how a node functions in an IE tree, we describe what each node represents and the various pieces of information that each node maintains. Each

node in the IE tree represents an AD subproblem with a fixed set of defenses. Each node maintains several distinct pieces of information.

- *Parent[n]* is the parent node of node n . Only the root node does not have a parent.
- *Children[n]* is a list of the child nodes for a node n . Child nodes have one additional fixed defense than the AD subproblem of a parent node.
- *Branches[n]* is a list of branches from a parent node n to child nodes. Branches represent the addition of a defense to either improve the objective function of the parent AD subproblem or change the parent node worst-case attack.
- *Objective[n]* is the objective function value of the AD subproblem solution of Equation (1b) or (1c) associated with node n .
- *Attack[n]* is the subset of edges of the worst-case attack associated with the AD subproblem solution of node n .
- *Budget[n]* is the amount of defense budget utilized at node n . We subtract the budget utilized at a node from the original total defense budget to determine the amount of remaining defense budget available for a child node. Selecting an existing edge for a fixed defense or adding a new edge as a fixed defense at a child node subtracts from the remaining defense budget inherited from a parent node. If the remaining defense budget is less than the cost of any defense, then the node is a leaf of the tree.
- *Inclusion[n]* is the list of fixed defense edges at node n . It is a subset of edges (i, j) that are fixed to the defended option (d_1). Inclusion lists are inherited from parent nodes and expanded. For example, $\hat{W}_{ijd_1}=1$.
- *Exclusion[n]* is the list of edges that cannot be defended at node n . It is a subset of edges (i, j) that are fixed to the undefended defense option (d_0). Exclusion lists are inherited from parent nodes and expanded. For example, $\hat{W}_{ijd_0}=1$.

2. Requirements of IE tree branches

A branch from a parent node in the IE tree represents one new defense to select in response to the worst-case attack of the subproblem contained within a parent node. A new defense could be the protection of an existing edge or the addition of a new edge to the network. In order for a branch to exist, three things must be true.

- There must be enough remaining defense budget to allow for a new defense to the system. In the test network, adding new edges as defenses costs more than defending existing edges. The new defense is represented by the formation of a new child node at the end of the branch.
- An unprotected or unadded piece of the network must exist to defend.
- The defense option that is considered by the branch must result in a change of the worst-case attack plan or an improvement in the subproblem instance objective function value from the parent node to the child node. If a change does not occur, the defense was not effective in prevention of the worst-case attack encountered in the parent node AD subproblem. If the AD subproblem objective function value gets worse from parent node to child node, the defender is not making an improvement to the system. Defenses that worsen the objective function are not added to the IE tree.

C. IE ALGORITHM FUNDAMENTALS

The strategy to optimally solve a DAD problem instance with IE is to restrict the defense budget to zero and then relax it one unit at a time. We start with an undefended system and then add defenses to AD sub problems one at a time until the defense budget is exhausted. Existing edge defenses can be added one by one if they either change the worst-case attack of the parent node or improve the parent node objective function value.

The optimal solution to the root node AD subproblem forms an upper bound on the master problem. The attack plan of the root node AD subproblem provides one of two methods to add branches to the next level of depth in an IE tree. Branches from the root node are added by defending each of the attacked edges from the root node worst-case attack one by one. Additional branches can also add a new edge as a defense, as long as it improves system performance or changes the worst-case attack.

Child nodes from each branch represent the network with the addition of either a defended or added edge. Each child node is a new AD subproblem instance because it has new fixed values for the defense variables (\hat{W}). The optimal solution to each of these new AD subproblem instances determines the child node objective function value and worst-case attack. Each AD subproblem instance optimal solution may also improve the upper bound on the DAD master problem instance. Each child node then becomes a parent node to determine the next new defense if the remaining defense budget is not

exhausted. Branches from each new parent node are formed in the same manner. This process repeats until the defense budget is expended. When a node is created with no remaining defense budget, that node is called a leaf node.

The inclusion list and objective function value of a node represents a feasible, but not necessarily optimal, solution to the original DAD problem. The node or nodes with the best objective function value represent the optimal solution to the original DAD problem instance. Figure 4 illustrates the beginning of an IE tree based on the test network with a defense budget of three units and an attack budget of two units. The system cost is shown as the objective value in the tree.

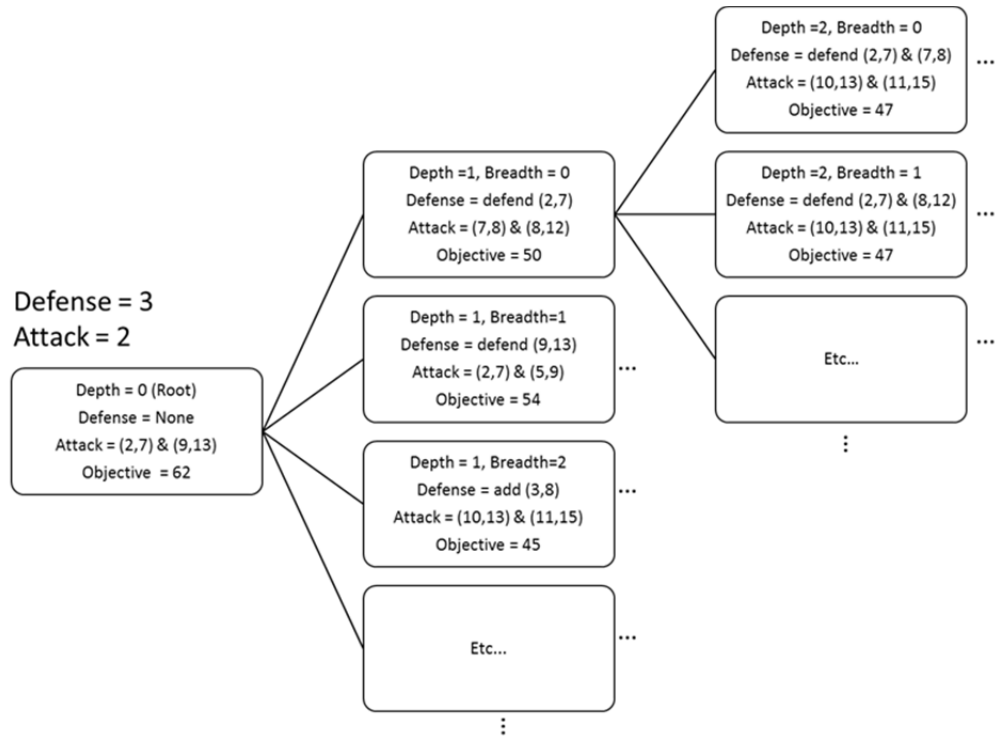


Figure 4. Portion of an Implicit Enumeration Tree

1. Skipping and Reconsideration of Potential Defenses

Branches form from a node based on defenses that can either improve the objective function value or change the worst-case attack. Only certain existing edges and certain new edges will meet the criteria for becoming an additional defense from a parent

node to a child node. Those edges that do not become defenses from a parent node to a child node are “skipped” from consideration for branches at a parent node. We use the term skipped because these edges are not permanently prohibited from being a defense. Each child node formed from a parent node via a branch represents a new AD subproblem instance. Each child node AD subproblem might feature a different worst-case attack that could be improved by defending an edge that was skipped by one of the previous parent nodes. This concept of skipping edges as enumerated defenses at an AD subproblem instance and then reconsidering them in subsequent AD subproblem instances is an essential feature of being able to efficiently construct an IE tree.

2. Avoiding Repeated Calculations

In order to improve the computation efficiency of our basic IE algorithm, we can perform a procedural check in the algorithm to prevent identical calculations from being performed more than once. Absent any oversight, an IE tree could have repeated inclusion lists at different nodes which would cause identical calculations. A repeated inclusion list will have all of the same elements, but that combination of defense elements may be ordered in a different permutation. Both of these nodes would have the same AD subproblem instance because the order of defense items does not affect the calculation. Repeat defenses at different nodes mean that the same AD subproblem instances must be computed multiple times. Calculating the same AD subproblem instance more than once is inefficient. The problem of repeating calculations is shown in a snippet of an IE tree of the test network with defense budget of two units and attack budget of four units in Figure 5. In the example, one node has an inclusion list to defend edges (10, 13) and (8, 12) and another node has an inclusion list of edges (8, 12) and (10, 13).

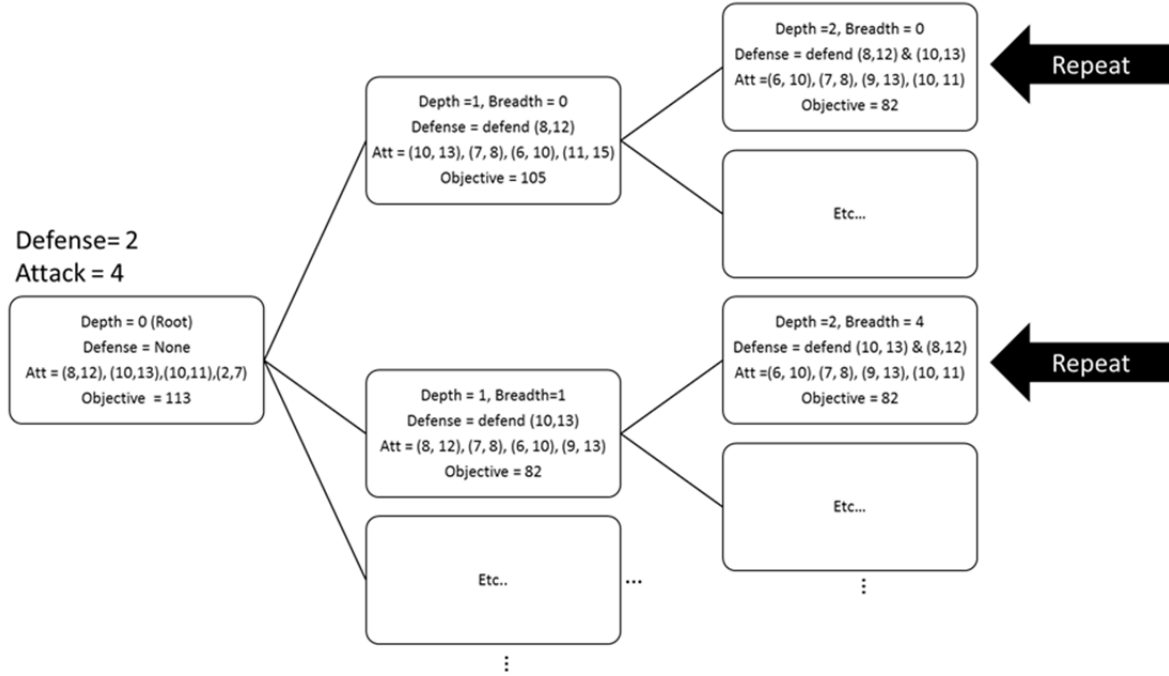


Figure 5. The Potential for Repeated Calculations in an IE Tree

The exclusion of repeated calculations can be thought of as a simple pruning mechanism for an IE tree. An exclusion list of previously considered defenses should be maintained for each node to prevent repeated identical AD subproblem instances. The exclusion list for a node is made by examining the defense chosen by the branches formed from a parent node. The first branch of a parent node does not exclude any new defenses, but the second branch prohibits the defense chosen by the first branch. The third branch prohibits the defenses chosen by the first two branches, and so on. Exclusion lists at a node will also include the exclusion lists of a parent node. The exclusion list can be stated as follows:

Exclusion rule: Once a branch is formed from a parent node that represents a particular defense, all subsequent branches from that parent node are prohibited from utilizing that defense. All descendent nodes from those subsequent branches are prohibited from selecting that defense as well.

For example, in Figure 5, if the first node at depth one defends edge (8, 12), then all of the other child nodes at depth one from the same parent node will be prohibited

from also defending edge (8, 12). When there are child nodes at a particular depth from different parent nodes, the exclusion list rule only applies to the child nodes from the same parent node. Exclusion lists are made from side to side for all of the branches from a particular parent node, with the beginning nodes not excluding anything, and following nodes excluding defenses that have been attempted by previous nodes at the same depth. Exclusion lists are passed from parent to child node and then appended with the excluded edges at the new depth. With the exclusion list mechanism, repeated calculations of the same AD subproblem instance are eliminated from the IE tree. Figure 6 illustrates how exclusion lists grow by showing a part of the IE tree for defense budget of three units and attack budget of two units. In the example, defense of edge (2, 7) is initially considered in depth level one, and excluded by all other nodes at that depth.

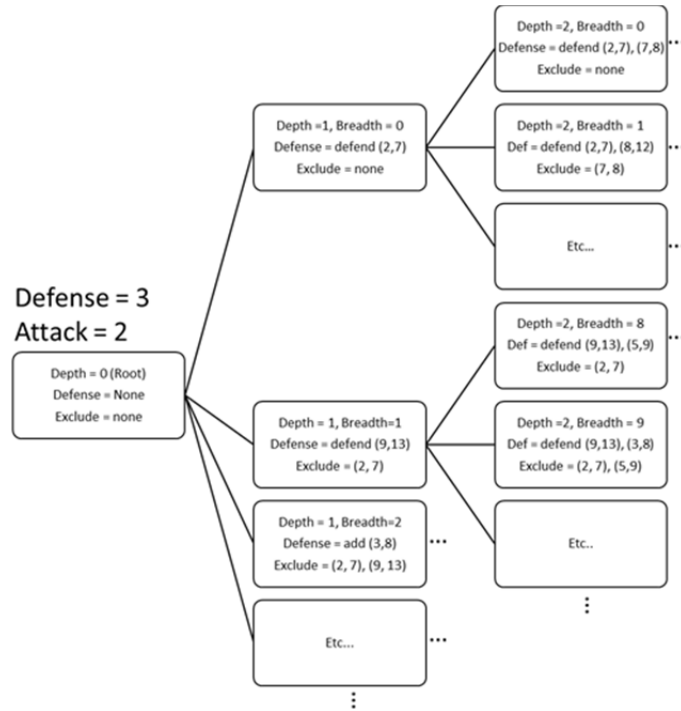


Figure 6. Exclusion Lists in an IE Tree

D. PRUNING OF NEW EDGES

The term new edge refers to an edge that has zero transport capacity (u) under the undefended defense option (d_0), and a transportation capacity (u) greater than zero when

defended (d_1). This type of edge can be thought of functioning in the network only if it is chosen as a defense. When considering how to evaluate the design of new edges as defenses to a network, we desire to develop a rule to exclude or prune some of the new edges from the tree when it is known that they will lead to suboptimal solutions of the DAD problem instance. In order to use an IE strategy for a network that has the capability of new edge design, a different approach is required than seen in Scaparra and Church (2008). The simplest method for an IE tree with new edges would be to consider every new edge for each node in the tree one at a time for network defense. But, the problem with considering each new edge at each node is that the total number of nodes in the enumeration tree will be very large. Without a rule for excluding some new edges at each new node, the resulting algorithm would perform inefficient explicit enumeration.

We also desire to develop a pruning rule that preserves the ability to implicitly enumerate defenses one at a time. Situations may exist where considering new edge defenses one at a time in an enumeration scheme may fail to identify the optimal defense. We show that situations exist where new edges could be considered in groups of two or more as a single defense. We desire to eliminate those situations from our enumeration strategy because it can result in a combinatorial explosion of the number of defenses that would need to be considered at each node.

1. A Simple Pruning Rule for New Edges Is Incorrect

Unfortunately, we cannot employ a simple pruning rule that restricts our consideration to those new components whose addition improves the performance of the network or reduces the impact of the worst-case attack. The following counterexample shows why a simple pruning rule does not always generate to the optimal defense.

Consider the simple five node network in Figure 7. The objective of the network is to move resources from source to destination at minimum cost. Define a transshipment node as a node where no source or demand for resources exists. A source node is black, a destination node is white, and a transshipment node is grey. Each destination node requires one resource, and the source node has exactly enough resources to fulfill all requirements. There is a cost of one unit to move one resource over any one edge. Edges

have unlimited capacity, and attacked edges are unusable. There is a penalty cost of 10 units if a destination node does not receive a resource. There is no cost for either using or not using a transshipment node. Existing edges are solid lines, and edges that may be added to the network are dashed. New edges can transport resources only if built. Nodes 1 through 4 are connected as shown, and node 5 is disconnected. Node 5 may be connected with the addition of one or more of the dashed line edges. The attacker has a budget of two units. The defender has a budget of two units, and defending an existing edge or building a new edge both cost one unit. Newly added edges to the network are invulnerable to attack.

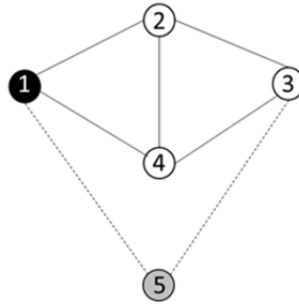


Figure 7. A Minimum Cost Flow Network with a Transshipment Node

Proposition 2–2: If new edges that could be added to a network are connected to a transshipment node, then the new edges cannot be considered for exclusion one at a time in an IE tree if no change to the worst-case attack or improvement in network performance is observed.

Proof of Proposition 2–2: Proof via contradiction. Assume to the contrary that new edges connected to transshipment nodes are considered one at a time for addition to the network. We use the five node network of Figure 5 as an example. The undefended attack plan for the root node would be to attack edges (1, 2) and (1, 4) which disconnects that source from the network and stops all flow of resources. The attack is a minimal edge-cut (Chartrand & Zhang, 2012, p. 116) of Figure 7. The undefended objective function would have value 30 because each destination node would each incur the penalty cost and no resources would flow over edges. Considering the addition of edge

(1, 5) by itself to the network as one defense would not have any effect, since no resources would reach their destination with this one addition. No resource would flow from node 1 to node 5 because there is a cost of 1 to use edge (1, 5) and no incentive to move one unit of resource to node 5. A similar argument applies to the addition of edge (3, 5). In both cases, these singular defenses would neither change the worst-case attack nor the network objective function value. If we consider new edges one at a time, both new edges would be excluded from the first set of branches to the IE tree. The optimal solution to this problem, though, is to add both edges (1, 5) and (3, 5) to the network simultaneously. Note that each source and destination node in the network with both new edges added is now 3-connected (Chartrand & Zhang, 2012, p. 117). Since each source and destination is 3-connected, the attacker cannot use two attacks to stop the flow of resources from any source to any destination. Notice that the resulting objective function of the defended network would improve to eight, but the worst-case attack would not change. Thus, a contradiction is reached because the optimal solution for the DAD problem instance can only be found in this network by considering both new edges simultaneously. Figure 8 shows the result, and the numbers above the edges represent the amount of resource flow on that edge. ▀

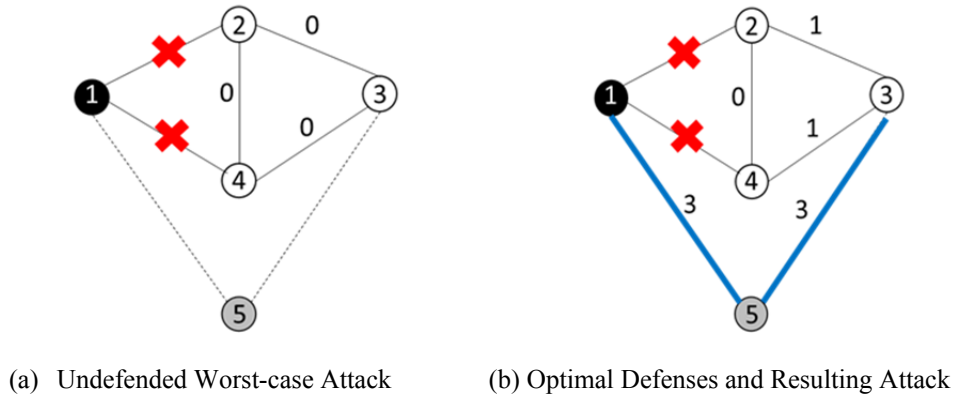


Figure 8. New Edge Consideration with Transshipment Nodes

We make another proposition about new edge exclusion in enumeration trees that does not rely on transshipment nodes with the network in Figure 9. In Figure 9, assume source node 1 has supply of three resources and source node 2 has a supply of one

resource. Both of the source nodes have exactly the amount of supply that is required by the destination nodes that they are connected to. Edges (1, 2) and (4, 6) do not currently exist, but they may be added as defenses. The defender has a defense budget of two units, and the attacker has a budget of one unit. Again, destination nodes are white, have a demand of one resource each and have a penalty cost of 10 if the demand is not satisfied. Solid lines are existing edges, and dashed lines are edges that could be built as defenses.

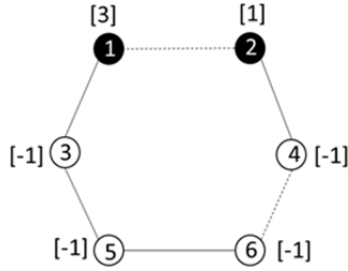


Figure 9. A Minimum Cost Flow Network with No Excess Supply

Proposition 2–3: If source nodes do not have supply in excess of the total demand at destination nodes, then no new edges that could be added to a network can be considered one at a time in an IE tree.

Proof of Proposition 2–3: Proof via counterexample. Assume to the contrary that edges could be excluded one at a time in a network that has nodes with no excess supply and consider Figure 9. In Figure 9, the worst-case attack would be to attack edge (1, 3) which cuts off flow to nodes 3, 5 and 6, resulting in an objective value of 31. The first best defense would be to defend edge (1, 3) to restore flow. Building edge (1,2) by itself would have no effect on either the worst-case attack or optimal value, since the worst-case attack would still occur at (1,3) and cut off the same nodes. Building edge (4, 6) by itself would also have no effect on either the worst-case attack or optimal value, since no resources could travel over the new edge. In this case, the IE tree would exclude both new edges from further consideration. The IE algorithm would next attempt to form the second defense. With edge (1, 3) defended in the first round of enumeration, the new worst-case attack would be edge (3, 5) to cut off flow to nodes 5 and 6. When

considering the second level of defenses, the IE tree algorithm would not be able to consider both new edges together, because the formation of the first layer of depth of the tree would have excluded both edges individually. The IE tree would only consider the new edges individually for the second defense, and they would not be optimal. However, the contradiction occurs because the optimal defense is to build both edges (1, 3) and (4, 6). The worst-case attack in response to this defense is unchanged. This defense would not be found by examining new edges for addition one at a time. When the two new edges are added simultaneously, the network topology becomes a 2-connected graph (Chartrand & Zhang, 2012, p. 115). Any single attack will result in a network that is still a 1-connected graph, and all resources can flow to all destination nodes. Figure 10 illustrates the proof. ▀

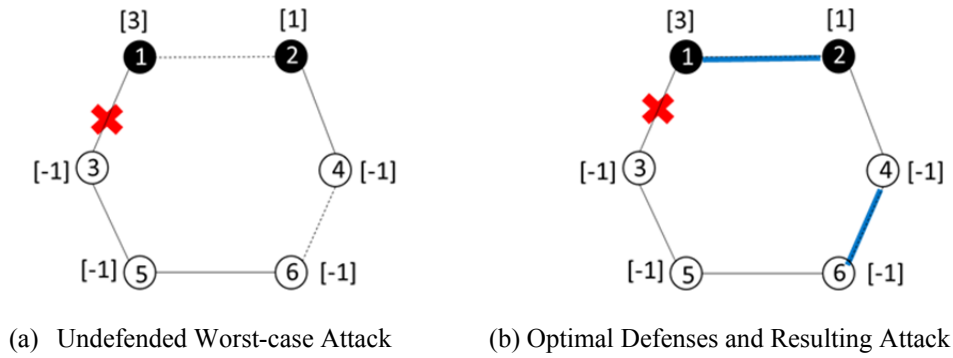


Figure 10. Lack of Supply in Excess of Total Demand

2. Excess Supply and Incentive

Propositions 2–2 and 2–3 provide ample evidence that the simple pruning rule is flawed. But, we still desire a decision rule for pruning some new edge branches in an IE tree. In order to develop a pruning rule that can incrementally add defenses one at a time, we must first introduce some additional terminology.

Define *excess supply* as a source node that has more resources available than it can deliver to all destination nodes that are connected to it. A source node is connected to a destination node by one or more paths of existing edges or previously added edges. This definition allows for the possibility that the network could be connected or disconnected

into multiple subnetworks. If an entire network is connected, excess supply means that the sum of all capacities of all source nodes is strictly greater than the sum of all demands of all destination nodes. Similarly for disconnected subnetworks, excess supply means the sum of all capacities of all source nodes in a subnetwork is strictly greater than the sum of all demands from all destination nodes in that subnetwork. Without excess supply, the situation in the previous proposition could occur.

Incentive is defined to mean that the cost to transport resources over a network from a source node to a destination node is less than the penalty cost for failing to fulfill demand at a destination node. Incentive assumes that a demand exists for a resource at a destination node. When incentive exists, the optimal solution will transport resources from source to destination because the cost of doing so is less than the penalty for not doing so. If incentive does not exist, then no reason to improve a network to transport supply to demand exists either.

Proposition 2–4: New edges must have an incentive to be built in order to be considered as defenses in an IE algorithm.

Proof of proposition 2–4. Assume to the contrary that edges did not need to have incentive to be considered in IE. For example, Figure 11 shows a supply node with more resources than demanded at nodes 2 and 3. The edge that connects nodes 1 and 2 has a cost less than the demand, so an incentive exists for resources to be transported to node 2. But, the new edge that connects nodes 2 and 3 would create a path from node 1 to node 3 that would have a traversal cost of eight units, which is more than the penalty of five units for not satisfying demand at node 3. In this bizarre case, the optimal solution would be for the resource to remain at the supply node and not travel to demand node 3, because it would cost more to transport the resource than to leave the destination unsatisfied. Optimally, edge (2, 3) would not be added to the network. The situation in Figure 11 can be avoided if the maximum cost of any acyclic path with new edges between source and destination nodes is less than the penalty at a destination node. If all edges have a unit cost, then the graph theoretic measure of “distance” between source and destination must be less than the penalty for failing to fulfill demand (Chartrand & Zhang, 2012, p. 15).▪

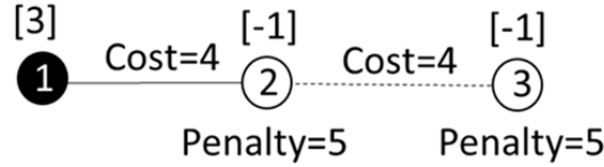


Figure 11. No Incentive Exists for New Edge to Demand Node

For the next proposition, consider Figure 12. Figure 12 contains three subnetworks that are each connected with existing edges. Subnetwork A only has demand nodes without any supply, so unmet demand exists. Subnetwork B has supply and demand nodes, but no excess supply exists. Subnetwork C has supply and demand nodes, and there is more supply available than demand. The interconnection edges between the subnetworks that are shown by the dashed lines between nodes 1 and 2 and 3 and 4 do not currently exist, but could be added as extra defenses. The penalty for unmet demand is significantly higher than the cost of transporting resources over the longest possible path that could interconnect any source and destination node across subnetworks. There is sufficient defense budget to add both new edges. Subnetwork A is a connected, but it penalizes the overall network objective function because of unmet demand. Subnetwork B is connected, and there is enough supply to satisfy demand, but nothing in excess. Subnetwork C is connected, and there is significant excess supply.

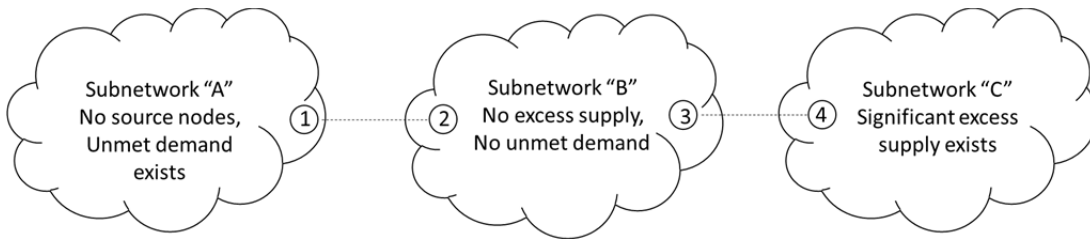


Figure 12. Disconnected Subnetworks with Unbuilt Connecting Edges

Proposition 2–5: Source nodes must have excess supply and new edges must have incentive in order to consider the incremental addition of new edges individually in an enumeration strategy.

Proof of proposition 2–5: Proof by counterexample. Assume to the contrary that network source nodes did not need to have excess supply and new edges did not need incentive and consider Figure 12. Suppose new edges were considered as network defenses individually for addition. In the first round of selecting defenses incrementally, new edge (1, 2) would not be added to the network because adding it alone does not have an effect on the network. Incentive exists for resources to travel to subnetwork A over edge (1, 2), but no extra resources exist in subnetwork B to utilize the new edge (1, 2). The lack of excess supply in subnetwork B would mean that if edge (1, 2) was utilized, then penalties would be incurred in subnetwork B to satisfy unmet demand in subnetwork A. The travel cost of resources over new edge (1, 2) would make the addition of edge (1, 2) by itself a suboptimal solution. Also in the first round of selecting defenses, new edge (3, 4) would not be added to the network by itself because it would have no effect either. Excess supply exists in subnetwork C, but no incentive exists over edge (3, 4) for the extra resources to only travel to subnetwork B. All demand in subnetwork B is already satisfied. In the second round of selecting defenses incrementally, edges (1, 2) and (3, 4) again would be considered individually and neither would be added for the same reasons as in the first round. The contradiction occurs because the optimal solution would be to add both edges to the network so that the excess supply in subnetwork C could travel over both new edges to satisfy the unmet demand in subnetwork A. Adding both edges simultaneously is optimal because the high penalty cost for unmet demand creates incentive for the excess supply in network C to be able to reach the unmet demand in subnetwork A. Since the optimal solution is a contradiction, the proposition must be true. Excess supply does not exist in subnetwork B and that makes the addition of either new edge appear to be inconsequential when they are considered individually.

The result from the example in Figure 12 is that when disconnected excess supply components and disconnected excess demand components are involved in a network, an incremental approach to enumeration can only be considered when a maximum distance of one potential edge can exist to connect the components. Figure 12 proves that if there are intermediate connections between excess supply subnetworks and unmet demand

subnetworks, then an incremental approach toward adding new edge defenses will not work.

The situation in Figure 12 could be avoided in at least two different ways. First, if the entire network was connected with other existing edges, then the situation would not arise. Second, if the source nodes in subnetwork B had excess supply, then incremental addition would be possible. Then, the new edge (1, 2) would be added to the network in the first round of enumeration. If the excess supply in subnetwork B was not enough to completely satisfy the unmet demand in subnetwork A, then the second round of enumeration would add the new edge (3, 4) to the network.

3. New-Edge Pruning Rule

Based on the propositions encountered in the previous sections, it is possible to write a pruning rule for new edges that could be added to a network as defenses. But, the analyst must ensure that a network meets the criteria specified in the rule. The analyst must ensure the network has source nodes with excess supply and that incentive can occur over new edges that could be added to the network.

Proposition 2–6 (Pruning rule for new edge defenses): Consider a network where new edges do not connect to transshipment nodes. If source nodes in the network have excess supply and new edges have an incentive to be added, then new edges considered in DAD IE must result in some type of improvement to the resulting system. Under these conditions, a new edge branch should be excluded from consideration at a particular node in a DAD network IE tree if it does not either change the worst-case attack or improve the system objective function value.

Proof of Proposition 2–6: Consider an arbitrary network of excess supply source nodes, edges, potential new edges, and destination nodes where source nodes have excess supply. Additionally, no new edges connect to transshipment nodes. An improvement to the network objective function will occur if resources can travel more efficiently from source to destination with the addition of the edge. In order for the new edge to impact the network, resources must be able utilize the new edge. We must determine if excess resources can reach the new edge addition. There are two cases to consider for the new

edge, either a path exists from an excess supply source node to a node that connects to the new edge, or a path does not exist.

Case 1, a path exists: Since we assume all source nodes have excess supply, then at least one source node has a path to at least one node that is connected to the new edge. At least one source node will be able to push its extra resources to one of the nodes on the end of the new edge. The node at the end of the new edge effectively becomes the equivalent of a new supply node because excess resources from the source node would be able to flow to it. The extra resources at the equivalent supply node are able to flow over the new edge to improve network objective function, or change a worst-case attack. Every new edge considered for inclusion will always behave like a source to source edge or a source to destination edge since we assume excess supply can reach the new edge. Since we assume no transshipment node at either end of the new edge, there is potential incentive for resource to flow on the new edge if it is added to the network. If the other end of the new edge is a destination node that already has its demand satisfied, the addition of the new edge may allow for a more efficient way to satisfy that demand, which would improve the system objective function. If the other end of the new edge is another source node, the new edge would create an equivalent “super source” node with combined resources that might result in more efficient flows. The improved efficiency of the system would be a reason to include the new edge in an enumeration tree. If the new edge does not provide a more efficient path to satisfy demand, then the addition of the new edge could be considered a waste and be pruned from the IE tree.

Figure 13 shows an example of the potential effects of excess supply source nodes. If source nodes 1 and 2 both have excess supply, then they will be able to deliver extra resources to the ends of the new edge (4, 6). Source node 1 will be able to turn node 6 into an equivalent source node since more than one resource can reach node 6. Likewise, source node 2 will be able to turn node 4 into an equivalent source node.

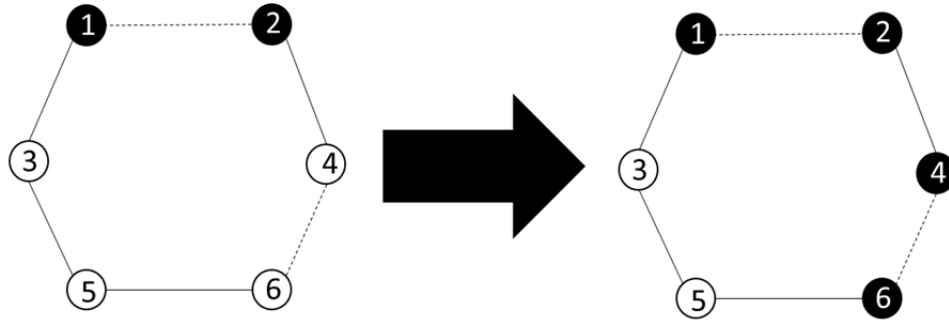


Figure 13. Equivalent Effect of Excess Supply Source Nodes

Figure 14 extends the example against an attack. Recall the optimal attack on this network was edge (1, 3), which cut off nodes 3, 5, and 6. That attack would not allow node 1 to move any excess supply anywhere. But, the excess supply of node 2 would turn node 4 into an equivalent supply node. Then, the inclusion of edge (4, 6) would restore flow because the equivalent supply node 4 would be able to send resources across edge (4, 6) to satisfy some or all of the previously cut off demand, depending on the amount of excess supply.

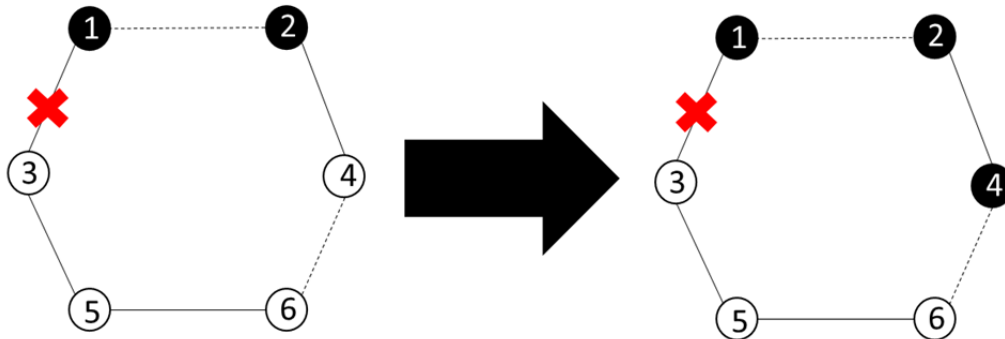


Figure 14. Attack with Excess Supply Source Nodes

Case 2, no path exists: If no excess supply source node has a path to either end of a new edge, then it would be impossible for any resources to flow to either end of the new edge. Consequently, no resources would be able to flow across that new edge if it was added to the network. In this case, the new edge would not serve any purpose in improving the efficient flow of resources within the system, and this new edge also

would not provide a reason for the worst-case attack to change. The addition of a new edge that cannot have flow would be a waste of a resource, and it could be pruned from an IE tree. Case 2 is illustrated in Figure 15, where excess supply node 1 does not have a path to the new edge between destination nodes 3 and 4. The branch representing the addition of edge (3, 4) could be pruned from the tree. ■

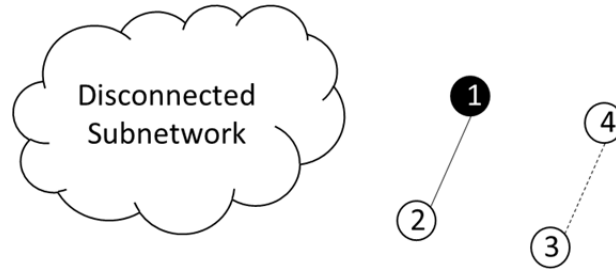


Figure 15. New Edge Unreachable by Excess Supply Source Nodes

4. Examples of New Edge Pruning

Since Proposition 2–6 provides us a rule to prune new edges in IE when the network has excess supply and incentive exists over new edges, we can examine some instances where pruning of a new edge may or may not occur.

Observation 2–7: If new edges do not connect to transshipment nodes, then the only possible types of new edges are source to source, source to destination or destination to destination.

New edges that connect source nodes to source nodes create a super supply node. The super supply node may be able to use the combined supply from both nodes to deliver resources to portions of the network that were previously unsatisfied. New edges that connect destination nodes to destination nodes create the equivalent of a super demand node, which would behave like a destination node with a much higher resource requirement. Behavior of super supply and super demand nodes in a network with excess supply can be easily seen by modifying a previous example to have excess supply. Consider Figure 16, with the worst-case attack shown. In Figure 16, the addition of edge (4, 6) creates an equivalent super demand of edges 3,4,5,6 to supply node 2. Since source

node 2 has excess supply, the addition of edge (4, 6) would be considered in the first layer of IE. But, even though node 2 has excess supply, it does not completely satisfy the demand. The second layer of enumeration to add the second defense would consider the addition of edge (1, 2) to the network which creates the equivalent of a super supply node. The new edge (1, 2) is added to the enumeration algorithm because an incentive exists somewhere in the network, specifically, the unsatisfied demand from node 3 that could exist on the path (1, 2, 4, 6, 5, 3). Since incentive exists, the edge (1, 2) is added to the network.

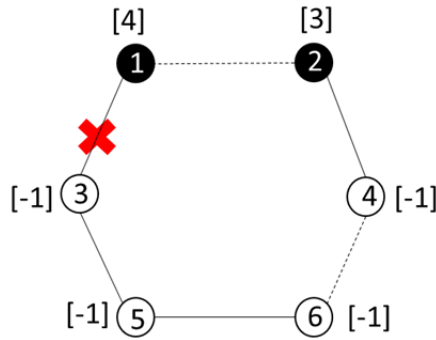


Figure 16. Worst-case Attack with Excess Supply Nodes

The source to destination edge is the last type of new edges that may be added to a network. There are two possibilities for new edges with source to destination nodes.

Possibility 1: The new edge is source to destination with previously unsatisfied demand. It should be obvious that addition of this new edge could improve the objective function value since the source node has extra resources available to send on the new edge to a destination that was previously penalizing the objective function for having unfulfilled demand. If the cost to transport between source and destination is higher than the penalty for unfulfilled demand, then incentive would not exist.

Possibility 2: Source to destination with previously satisfied demand. There are four types of situations that could occur.

Situation 2a: If the cost of moving resources to the destination node can be improved by adding the new edge to the network, then that new edge would be a candidate for addition.

Situation 2b: If the resource that is already satisfying the demand at the destination could be better utilized by satisfying some other demand on the network, then addition of the new edge will be an improvement and a candidate for addition.

Situation 2c: If the addition of the new edge does not change the movement of previous resources, then does the satisfied destination node appear to be the equivalent to a super supply node with the addition of this new edge? The satisfied demand node could look like an equivalent supply node if there is another existing incentive in the network for the edge to be added. While the satisfied demand node itself may not present an incentive for the edge to be added, there may be another efficiency in a different part of the network that will make the satisfied demand node appear to be the equivalent of another supply node.

Situation 2d: If the addition of the new edge does not change the movement of resources and there is no other existing incentive in the network for the edge to be added, then the candidate edge will not affect a change in the efficiency of the network nor change the worst-case attack. This new edge should be temporarily excluded from consideration in an IE algorithm.

In order to illustrate how new edge pruning works on the test network, consider Figure 17. Figure 17 is a portion of the IE tree on the test network with a defense budget of four units and an attack budget of one unit. The black arrows in Figure 17 show the defenses that could be pruned from the tree. The new edges labeled for pruning have the exact same worst-case attack and objective function value as the parent node. In this case, these defenses do not make any improvement to the function of the system, in terms of system cost or worst-case attack. These potential defenses are ruled to be a waste of defenses since improvement in objective or change in worst-case attack did not occur, and can therefore be pruned from the tree. These edges will be reconsidered as additional defenses in different child nodes later on as the IE strategy continues.

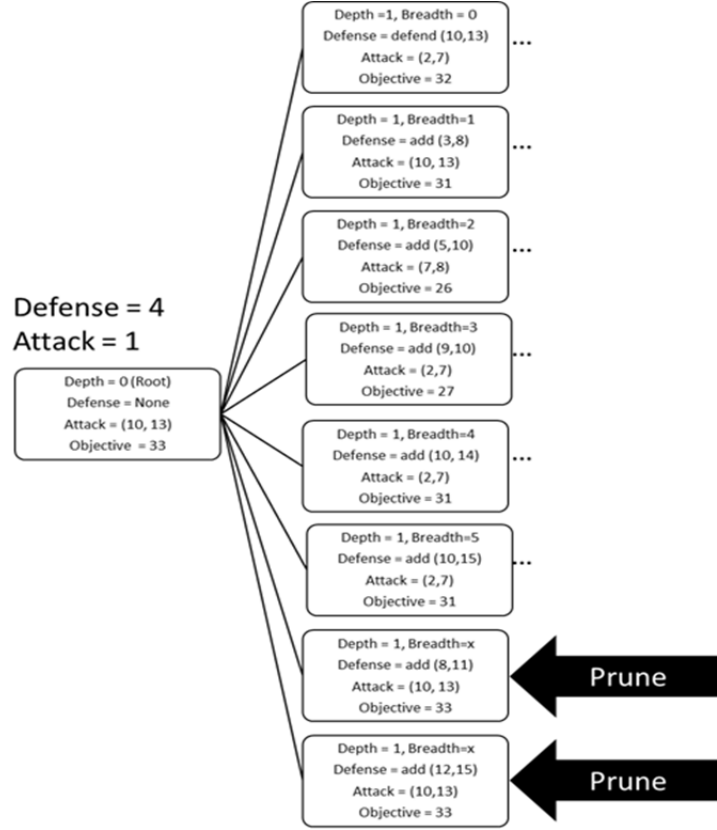


Figure 17. Pruning New Edges in an IE Tree

E. UPPER AND LOWER BOUNDS IN IE

As the IE tree is explored, we can update the upper bound on the overall DAD problem solution when each node AD subproblem instance is solved. Every node represents an AD subproblem instance that is a feasible solution to the master DAD problem instance. Each time a feasible solution is computed, it can be compared to the best known feasible solution found on the tree so far. If the new node AD subproblem objective function value is better than the best known objective function value, then the upper bound on the master DAD problem can be updated. By updating the upper bound on the DAD problem with each new node, the best known solution can be retrieved at any point during the execution of the algorithm.

A simple global lower bound for the DAD problem can also be computed. Consider a different restriction to the DAD problem, where instead of manipulating the defense budget, we instead choose to fix the attack budget. If the attack budget is fixed at

zero units, then we reduce a tri-level optimization problem to a different bi-level optimization problem. When there are zero attacks to consider, we have reduced the DAD problem to a “defender-operator” or “defender-defender” (DD) problem. The DD problem is also recognized as an example of the well-known network design problem (Magnanti & Wong, 1984). The DD problem is a mixed integer linear program since the only objective is minimization, as the attack variables are all fixed to zero.

$$\min_{W \in \Psi} \min_{Y \in \Upsilon(W)} f(W, \hat{X}, Y) \quad (2.2)$$

The formulation in Equation (2) is simply a restatement of the original DAD formulation (1a), except that all of the attack variables (X) have fixed values of zero. The formulation of the DD minimum cost flow problem has all attack variables (X) fixed at zero, and removed from the problem. The system of Equations of (2.2a-h) are modified from the original DAD minimum cost flow model in Chapter I, with all of the attack variables removed from the formulation.

Defender-Defender (DD) Minimum Cost Flow Formulation:

$$\min_{W_{ijd}} \min_{S_n, Y_{ijd}} \sum_{d \in D} \sum_{(i,j) \in E} c_{ijd} \cdot Y_{ijd} + c_{jid} \cdot Y_{jid} + \sum_{n \in N} S_n \cdot p_n, \quad (2.2a)$$

Subject to:

$$\sum_{d \in D} \sum_{(n,j) \in A} Y_{njd} - \sum_{d \in D} \sum_{(i,n) \in A} Y_{ind} - S_n \leq demand_n \quad \forall n \in N, \quad (2.2b)$$

$$\sum_{(i,j) \in E} Y_{ijd} + Y_{jid} \leq u_{ijd} \cdot W_{ijd} \quad \forall (i,j) \in E, d \in D, \quad (2.2c)$$

$$\sum_{d \in D: d \neq d_0} \sum_{(i,j) \in E} h_{ijd} \cdot W_{ijd} \leq defense_budget, \quad (2.2d)$$

$$\sum_{d \in D} W_{ijd} = 1 \quad \forall (i,j) \in E, \quad (2.2e)$$

$$Y_{ijd} \geq 0 \quad \forall (i,j) \in A, d \in D, \quad (2.2f)$$

$$S_n \geq 0 \quad \forall n \in N, \quad (2.2g)$$

$$W_{ijd} \in \{0,1\} \quad \forall (i,j) \in E, d \in D. \quad (2.2h)$$

The global lower bound obtained from the DD subproblem instance has limited use. The global lower bound represents the best operation of the system that a defender

could possibly experience when choosing defenses, since no attacks occur. The global lower bound provided by the DD subproblem gives the analyst a best case standard to measure the effectiveness of the best defenses against a worst-case attack. Calculation of the global lower bound is not required to produce an IE tree.

A simple example of updating the bounds can be seen in Figure 18, a DAD problem instance of the test network with a defense budget of three units and an attack budget of two units. First, the DD network design subproblem instance of no attacks gives us a global lower bound of 22 units. We know that no feasible solution of the DAD problem instance can be less than 22 units. Next, the root node AD subproblem instance of no fixed defenses gives an objective function value of 62 units, which becomes our initial upper bound. The first AD subproblem instance at tree depth of one gives an objective function value of 50 units, which updates the upper bound. The second subproblem instance at depth of one gives an objective function value of 54 units. But, this value is not an improvement over the previous best known value, so the upper bound is not updated. The next AD subproblem instance node at depth one has an objective function value of 45 units, which updates the upper bound. The remainder of the tree is not shown in Figure 18 and 45 units is the optimal solution found at multiple nodes.

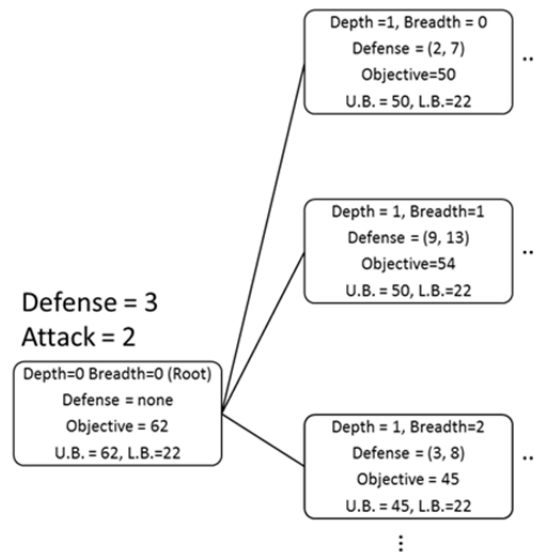


Figure 18. Upper and Lower Bounds in an IE Tree

F. IMPLICIT ENUMERATION ALGORITHM

The IE algorithm for the DAD problem has been implemented in the Python programming language with external calls to GAMS optimization software with the IBM CPLEX solver. We present the pseudo-code of IE to explain our algorithm in detail. Our algorithm builds and traverses the IE tree using a breadth first search strategy, and can be modified for other strategies.

Main routine:

Inputs:

- Network (supply nodes, destination nodes and costs & capacities of edges).
- Defense budget and attack budget for DAD problem instance.
- Set of new edges that can be added to the network satisfying three tests:
 - New edges may not connect to transshipment nodes.
 - Source nodes must have excess supply.
 - Incentive must exist for new edge to be added to network.

Outputs:

- DAD solution(s) = Best node(s) objective function value.
- DAD defense plan(s) = Inclusion list(s) with best objective function value.
- DAD attack plan(s) = Attack plan(s) of node(s) with best objective function value.

Algorithm:

- Solve AD subproblem with zero defense budget for root node.
- Build tree with breadth first search strategy:
- Relax defense budget by one defense to increase depth of tree.
- While** current depth \leq defense budget:
 - For** each node in breadth of depth:
 - Make children** with parent node data.
 - Relax defense budget by one defense to increase depth of tree.

Note: Multiple defense and/or attack plans may exist at optimal DAD solution value.

Make children subroutine:

Input: Parent node data (inclusion list, defense budget remaining, attack plan).

Output: All child nodes with all calculations performed.

Algorithm:

- Two options to create child nodes:
 - (First option: make children by defending attacked edges of parent node.)
- If** defense budget remaining at current node \geq edge defense cost:
 - For** every edge in current node attack plan:
 - Add new child node to IE tree.
 - Create node exclusion list with **make exclusions**.
 - Inclusion list = inclusion list of parent node + edge.
 - If** inclusion list is in node exclusion list:
 - Delete new node and new inclusion list.

If inclusion list **not** in node exclusion list:
 Solve AD subproblem for new node.
 Update upper bound with **update solution**.
 Set parent node of new node.
 Add new node to list of children of parent node.
 Decrease budget remaining at new node by cost of defense.
 (Second option: Make children of current node by adding new edges to network)
If defense budget remaining at current node \geq new edge addition cost:
 If any new edge in parent node inclusion list **OR** in parent node exclusion list:
 Delete edge from list of new edges.
 For each new edge in list of new edges make a candidate node:
 Candidate inclusion list = parent node inclusion list + new edge.
 Solve AD subproblem for candidate node.
 If candidate objective function value < parent objective function value **OR**
 If candidate attack plan is different than parent node attack plan:
 Candidate node is added as a new child node to IE tree.
 Create node exclusions with **make exclusions**.
 Add a new branch from parent node with label of new edge.
 Update upper bound with **update solution**.
 Decrease budget remaining of new node by cost of adding edge.

Make exclusions subroutine:

Input: Exclusion list of parent node.

Output: Exclusion list of child node.

Algorithm:

Set child node exclusion list to parent node exclusion list
For each currently existing branch of parent node:
 Add edge label of branch to current node exclusion list.

Update solutions subroutine:

Inputs:

Best known objective function value.
 List of tree node locations of the best known value.
 Current node objective function value.

Outputs:

Best known objective function value.
 List of node locations with that value.

Algorithm:

If current node objective function value is better than best known value:
 Set best known solution to current node objective function value.
 Set location of best solution to current node depth and breadth.
If current node objective function value is equal to best known solution:
 Append current node depth and breadth to list of locations of best known value.

1. Depth First Search Algorithm Option

Our IE algorithm is set up for the breadth first search tree exploration. If the analyst desires to perform a depth first search tree exploration, then the “build tree” step of the master algorithm can be replaced with this alternate version that uses a last in, first out stack data structure to explore the tree. Each node also has a new data element we call “visited” to check if the node has been visited before.

(Alternate) Build tree with depth first search exploration strategy:

Push root node to top of stack.

Root node visited = False.

While stack is not empty:

Current node = Pop node from top of stack.

Current node visited = True.

If defense budget remaining at current node \geq defense cost:

Call “make children” subroutine for current node.

For each child node of current node:

Push child node to top of stack.

Set visit for child node = False.

2. Algorithm Testing and Performance

The IE algorithm is implemented on the test network. Results were checked against results obtained with the DAD problem nested decomposition algorithm and the DAD problem dual ILP with decomposition algorithm from Chapter I. Results were double checked with the solutions of the DAD problem on the test network found in Alderson et al. (2015). Solution times in seconds for DAD problem instances are shown for both decomposition algorithms and the IE algorithm in Table 3.

Table 3. Comparison of Solution Times for Different DAD Solution Methods

Problem Instance		Solution Time (Seconds)			Multiple Optimal Solution
Defense Budget	Attack Budget	Nested Decomposition	Dual ILP + Decomposition	Implicit Enumeration	
0	1	4.8	0.7	0.5	NO
0	2	5.4	0.7	0.4	NO
0	3	7.5	0.7	0.4	NO
0	4	4.2	0.8	0.4	NO
0	5	2.5	0.7	0.4	NO
1	1	10.0	1.7	0.8	NO
1	2	11.2	1.8	2.3	NO
1	3	12.4	2.6	1.6	YES
1	4	22.1	2.3	2.0	NO
1	5	12.8	1.4	2.5	NO
2	1	14.0	2.4	4.9	NO
2	2	16.8	4.0	5.2	NO
2	3	48.8	4.3	8.5	NO
2	4	64.5	8.7	10.5	YES
2	5	58.0	11.0	13.9	NO
3	1	11.9	2.7	9.3	NO
3	2	21.0	2.7	15.0	YES
3	3	40.1	11.8	35.1	YES
3	4	110.9	14.3	48.5	YES
3	5	147.4	40.1	58.2	NO
4	1	13.1	2.0	26.9	YES
4	2	41.6	6.6	50.4	NO
4	3	70.2	10.5	121.8	NO
4	4	156.1	27.8	188.2	NO
4	5	247.7	61.0	238.3	YES
5	1	8.1	2.2	58.7	NO
5	2	51.4	8.7	144.1	YES
5	3	106.2	22.4	440.7	NO
5	4	197.4	37.6	554.4	NO
5	5	322.1	135.9	826.1	YES
Totals:		1840.2	430.1	2869.9	9

In general, Table 3 shows that the solution speed of the IE algorithm is roughly equal to or slightly slower than either decomposition approach for finding an optimal solution to the DAD problem on the test network. IE is the fastest algorithm only in the

case of zero defense budget. Table 3 shows that all of the algorithms take longer to solve a DAD problem instance when the size of the defense budget increases. But, IE becomes much slower than either decomposition method when the defense budget is five units.

However, there is one performance aspect of IE that makes it superior to both decomposition solution methods. IE is able to find many multiple optimal solutions at once, which is something that the nested decomposition algorithm cannot do. Table 3 shows the problem instances where multiple equivalent optimal solutions exist. Both nested decomposition methods may be faster than IE, but they do not know if multiple equivalent optimal solutions exist, and they do not provide equivalent optimal solutions either. IE can inform the analyst of equivalent optimal solutions when they exist. If the analyst desires to find as many equivalent optimal solutions as possible to a DAD problem instance, IE is a good choice. IE also shows the analyst that when multiple optimal solutions exist, they need not always occur with a completely utilized defense budget. Similarly, an analyst could make a minor modification to the algorithm in order to use IE to find all solutions within a fixed percentage of the optimal solution. The analyst could specify a percentage of optimality desired for all solutions, and the IE algorithm would be able to identify all of these solutions.

Proposition 2–8: When multiple optimal solutions are possible in a DAD problem instance, optimal solutions do not always occur at leaf nodes of the IE tree.

Proof of proposition 2–8: Proof via counterexample. Assume to the contrary that optimal solutions can only occur at leaf nodes in an IE tree. Consider the test network with a defense budget of three units and an attack budget of two units. The optimal solution to the DAD problem is 45 units. There are seven nodes on the IE tree with an objective function value of 45, and one of those nodes is not a leaf node. When only one defense is fixed by adding edge (3, 8), the optimal objective function value is 45 units, but the defense budget is not exhausted. The worst-case attack are edges (10, 13) and (11, 15). As it turns out, defending each of these attacked edges one at a time will create child nodes with defenses of (3, 8) with (10,13) and (3,8) with (11, 15). Both of these child nodes also result in optimal objective function values of 45 units. These child nodes are not pruned from the tree because the worst-case attack changes from parent node to child

node. In this case, the parent and child nodes in a subtree all share the same optimal objective function value. Figure 19 shows the multiple optimal solutions from a test network instance as green nodes. Thus, it is proven via counterexample that optimal solutions need not only exist at leaf nodes of an IE tree. ▀



Figure 19. IE Tree with Multiple Optimal Solutions

G. ENUMERATION SUMMARY AND CONCLUSIONS

The IE algorithm works by restricting and subsequently relaxing the defense budget of the original tri-level DAD problem in order to generate easier to calculate bi-level AD sub problems that can help us find all reasonable choices of defenses. The restriction of the problem is performed by fixing the values of the defense variables (\hat{W}) one at a time. At the root node, the defender is restricted in having zero defense budget, which is the initial AD sub problem. At the first level of depth, the restriction of the root node problem is eased slightly because the defender is allowed one defense. At depth one

in the IE tree, the defender can choose only one defense that either improves the objective function or changes the worst-case attack of the parent node AD sub problem. This rule for node development at the next depth effectively prunes all other possible defenses that would be suboptimal. In this manner, explicit enumeration is avoided because all possible choices of defenses are not calculated in the tree. All of the AD subproblem instances in the IE tree are also easier to calculate than the original DAD problem instance. The methodology of preventing repeat calculations also helps to prune repeat calculations from the enumeration tree. The process of slight easing of the restriction on fixing the defense variables will continue throughout the formation of the IE tree until the defense budget is exhausted. Additionally, the formation of each new node performs a check for the updating of the best known feasible solution. This feature allows the analyst to be able to pause the algorithm and get the best known feasible solution at any time.

Proposition 2–9: The IE algorithm is an exact algorithm that produces an optimal solution to a DAD problem instance. If the IE tree is completely explored, then one or more equivalent optimal solutions will be found to the DAD problem instance.

Proof of proposition 2–9: An enumeration strategy lists all possible solutions to a problem instance. IE discards feasible combinations of decision variables that cannot lead to an optimal solution. In our implementation of IE for a DAD problem instance, we compute only those feasible combinations of defenses to worst-case attacks that can improve the objective function value or change the worst-case attack from a previous AD subproblem instance. The complete IE tree will list all feasible combinations of defenses that can either improve the objective function value or change worst-case attacks. The combinations of defenses that do not do either of those two things cannot lead to an optimal solution to the DAD problem instance. If the tree exploration is prematurely halted before completing the IE algorithm, an optimal solution might be found, but it also may exist in a portion of the tree that had not yet been completely explored.▪

However, IE does have a shortcoming. The size of an enumeration tree depends on the magnitude of the defense and attack budgets, the costs of defenses and attacks to those budgets, and the cardinality of the set of new edges that may be added to the network. The depth of the tree is equal to the number of defenses selected in AD sub

problems. The breadth of the tree at any particular depth depends upon the number of attacks permitted by the attacker budget and the cardinality of the set of new edges. Thus, the shortcoming of IE is that the tree size can become incredibly large when either the defense budget, attack budget, or number of network design elements becomes large.

In the case where there are no new edges that can be added to the network, the maximum size of the tree is straightforward to compute. The breadth of each level of the tree is the number of attacks raised to the exponent of the level of the tree. Thus, with three defenses and two attacks, the maximum size of the tree is 15 nodes. The maximum size is computed as follows: level zero consists of $2^0=1$ node; level one has $2^1=2$ nodes; level two has $2^2=4$ nodes; level three contains $2^3=8$ nodes. The maximum number of enumeration nodes in this tree is sum of all levels, $2^0 + 2^1 + 2^2 + 2^3 = 15$ nodes. The tree gets really big fast. If there were 5 attacks and 5 defenses, the maximum number of enumeration nodes would be $5^0 + 5^1 + 5^2 + 5^3 + 5^4 + 5^5 = 3906$ enumeration nodes, each containing an AD mixed integer program (MIP) subproblem. However, the maximum size of the tree is not the actual size of the tree when exclusion lists are used to prevent repeated calculations. Nevertheless, we observe the problematic issue with IE is the number of MIPs to solve grows exponentially with the number of defenses allowed by the defense budget. Adding new edges into consideration makes the maximum number of nodes even larger.

IE may not always be the fastest way to the optimal solution to a DAD problem instance, but this approach does have the unique capability of quickly identifying multiple optimal solutions. If an analyst needs to consider all equivalent optimal solutions in order to make a list of defenses to choose, IE may be a good choice.

THIS PAGE INTENTIONALLY LEFT BLANK

III. NESTED DEFENSES IN THE DAD PROBLEM

A. INTRODUCTION

When faced with uncertain budgets and pressure to make good decisions quickly after defense budgets become known, military and civilian decision makers sometimes make use of a prioritized list of assets to defend in a system. Given a prioritized list and a realization of defense budget uncertainty, a decision maker may be inclined to choose to defend as many items from the top of the prioritized list downwards as he or she can afford. Prioritized lists can simplify decision making in uncertain budget environments, but this simplicity comes at the cost of optimality. Requiring the set of defenses to be presented as a prioritized list is equivalent to requiring monotonicity on the sets of defended assets as budgets increase.

Uncertain budgets can refer to either an unknown defense budget, or an unknown attack budget of an adversary, or both. In the unknown defense budget scenario, decisions about which defenses to make to a system may need to be finalized before a defense budget is known. In the unknown attack budget situation, the defenders of a system may have to choose their defenses without any intelligence about the expected number of enemy attacks on a system. In the circumstance of complete budget uncertainty, the analyst knows nothing about the defense budget and the budget of their attacking enemy, but he or she must prioritize which defenses are best for a system.

A nested defense means that for any pair of defender-attacker-defender (DAD) problem instances with different defense and/or attack budgets that the defense chosen with the smaller budget is a subset of the defenses chosen for a larger budget (Nehme & Morton, 2010). Nested defenses are a monotonic sequence of sets. A closely related term is “prioritization”, which is defined as the process of arranging choices in a list by their priority. In the context of the DAD problem, prioritization of defenses is the ability to rank the relative ability of each different defense to positively affect the objective function value against a worst-case attack. Koc and Morton state, “prioritization involves optimally placing activities into a priority list before the uncertainty is revealed, and, after

realizing the uncertainty, making an activity selection consistent with the priority list” (2015, p. 587). An example of a nested defense is a prioritized list of the best defense against any attack, second best defense, and so on. In the real-world problem of infrastructure defense, prioritization is a logical choice for planning against a wide range of possible defense and attack scenarios. “Prioritization is of interest when some problem parameters are random and we must commit to a ranking of the activities before these parameters are realized” (Koc & Morton, 2015, p. 586).

Ranking the relative importance of defenses of a network into a prioritized list may seem to be a natural method for planning to defend a system when faced with uncertain budgets, but it is not optimal most of the time. Savage et al. (2006) refer to the suboptimal nature of prioritized lists as the “risk of ranking.” The authors note that “It is common when choosing a portfolio of capital investment projects to rank them from best to worst, then start at the top of the list and go down until the budget is exhausted. This flies in the face of modern portfolio theory which is based on the interdependence of investments” (2006, p. 22). In the case of the DAD problem, the choice of defenses represents the capital improvement projects mentioned by these authors. A prioritized list can simplify decision making, but it can lead to a suboptimal solution for any particular budget chosen. It is often the case that the one best defense of a system is not included in the set of the two best defenses, and so on. Brown, Carlyle, Salmeron and Wood. found that “a prioritized list of defended assets has a serious flaw... Such a list creates a “preferred set” of “ $n+1$ ” assets by adding one asset to the preferred size “ n ”. But, we know that an optimal set of size “ n ” and an optimal set of size “ $n+1$ ” may have nothing in common” (2006, p. 531).

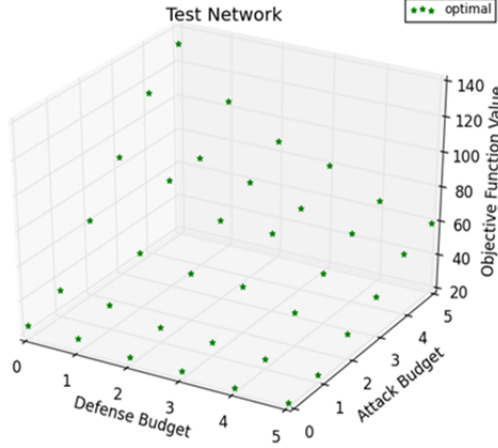
While it is known that nested defenses are often suboptimal solutions, an abundance of research on the impacts of requiring nested defenses for bi-level DA models or tri-level DAD models does not exist. “In spite of its common use in practice, prioritization has received little attention in the academic literature.” (Koc & Morton, 2015, p. 587). One discovery is that the requirements for a naturally occurring nested defense in an optimization problem have been determined. Nehme and Morton show “the nestedness property hinges on supermodularity of the objective function and

submodularity of the [constraints]” (2010, sec. 3). However, these authors also conclude that many real-world problems do not exhibit these qualities.

DAD models that include nested defenses with uncertain budget levels is a parametric programming problem. Bertsimas and Tsitsiklis describe parametric programming as a systematic procedure for obtaining objective function values for all values of a parameter (1997, p. 218). As a parametric programming problem, our DAD model places the uncertain parameter on the right hand side of the defense and/or attack budget constraints. When the defense and/or attack budget is uncertain, the analyst takes into account the entire range of budget scenarios in order to find a solution that contains common defenses.

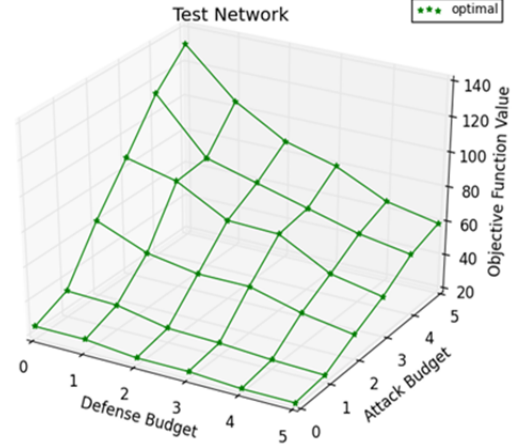
This chapter examines how much of an impact to optimality is incurred when nested defenses are required in a DAD problem instance with unknown defense and/or attack budgets. This chapter does not consider the idea of nested attacks by an adversary. We compare nested defense solutions against the optimal defense solutions to DAD problem instances in order to measure the effects of nesting. We use the DAD minimum cost flow problem formulation as well as the network of Alderson et al. (2015) described in Chapter I to illustrate nested defenses. Procedures to obtain optimal solutions to DAD minimum cost flow problem instances were displayed in Chapter I. In Figure 20, we plot the optimal solutions to the test network DAD problem for different defense and attack budgets. Figure 20, graph ‘a’ depicts the set of optimal solutions of the DAD problem instances for the test network by varying the defense and attack budgets on two axes, and displaying the corresponding optimal objective function value on the third axis. Each green star represents a DAD problem instance of a specific defense budget and attack budget. The optimal objective function values for DAD problem instances are a set of discrete points, since fractional values for defense budgets and attack budgets are meaningless. Figure 20, graph ‘b’ depicts objective function value points connected with line segments. The line segments are for the illustrative purpose of enhancing the three dimensional effect of the plot, and are not meant to imply the existence of a continuous surface.

DAD Problem Instance Optimal Solutions



(a) Test Network Optimal Solutions

DAD Problem Instance Optimal Solutions



(b) Optimal Solutions and Connecting Lines

Figure 20. Three Dimensional Plots of Test Network Optimal Solutions

B. IMPLEMENTATION OF NESTED DEFENSES

In order to implement nesting of defenses, a new set has to be introduced that represents budget scenarios. Budget scenarios allow for a parametric analysis of varying one or both budgets in the DAD model. We use *omega* to define a specific budget scenario (ω) and the set of all budget scenarios (Ω). The decision variables must carry a new superscript that identifies the budget scenario.

Additional Sets:

$$\omega \in \Omega$$

Parameterized unknown budget scenarios

There are at least two ways to incorporate the requirement for nested defenses in the DAD model. We investigate both approaches. The first method uses a block of constraints to force the defenses utilized in one budget scenario into the next scenario with a different value for the defense and/or attack budgets. The second method is to solve each problem instance separately in the order of the nesting strategy. We fix some of the defense variable values (W) from the previous problem instance for the next problem instance. We use the hat symbol ($\hat{}$) for a defense variable with a fixed value.

1. Nesting as Constraint Equations

Implementation of nesting constraints in a parametric linear program allows us to examine all budget scenarios simultaneously. A new problem formulation objective function includes summation over all of the budget scenarios. The formulation of the DAD minimum cost flow problem is similar, but it now includes the new budget scenario superscript on all of the decision variables and budget data. There is a new constraint at the end of the formulation that depends upon the type of nesting strategy used. If the nesting strategy begins at the minimum budget, then Inequality (3.1a) is used. If the nesting strategy begins with maximum budget, then Inequality (3.1b) is used. If the nesting strategy begins at an intermediate budget, then a feasible combination of Inequalities (3.1a) and (3.1b) are used.

Nesting Constraints

$$W_{ijd}^{\omega} \leq W_{ijd}^{\omega+1} \quad \forall (i, j) \in E, d \in D, \omega \in \Omega. \quad (3.1a)$$

$$W_{ijd}^{\omega} \geq W_{ijd}^{\omega+1} \quad \forall (i, j) \in E, d \in D, \omega \in \Omega. \quad (3.1b)$$

2. Nesting by Fixing Variables

A second method to implement nesting of defenses is to “fix” some of the defense variables to one or zero, depending on the starting instance of the nesting algorithm. We start with one DAD problem scenario of a specific defense and attack budget and solve it optimally. Next, take the defenses from the original instance as fixed values to a second DAD problem instance with the parameterized budget varied by one unit. Solve the second instance with the additional constraint optimally to obtain defenses in the second instance that are also in the first problem instance. This process repeats until all possible budget scenarios are solved and nested defenses are obtained. Some of the defense decision variables (W) change to fixed values (\hat{W}), depending on the direction of the nesting strategy being used. Each budget scenario problem instance must be solved, one at a time, in the order of the nesting strategy.

a. *Nesting Rule for Scenarios with Increasing Budgets*

First, the optimal solution for the starting point budget scenario is obtained. The optimal solution of the next budget scenario must contain all of the defenses chosen in the previous problem instance, and it may include another defense if an increase in defense budget permits another defense. The defenses chosen (W) are changed from decision variables into fixed values of one. Next, a new problem instance is formed with the next scenario of increased defense and/or attack budgets. Nesting defenses under increasing budget scenarios can be called “fixing defenses to one” since as defenses are chosen in smaller budget scenarios, the value of their decision variable is fixed to one for larger budget scenarios. Equation (3.1c) summarizes the fixing of defense variables when a nesting strategy increases budgets.

$$\hat{W}_{ijd}^{\omega+1} = 1 \quad \forall W_{ijd}^{\omega} : W_{ijd}^{\omega} = 1. \quad (3.1c)$$

b. *Nesting Rule for Scenarios with Decreasing Defense Budgets*

We start with the optimal solution to a DAD instance with a maximum defense and/or attack budget. The next problem instance has a reduction in the defense budget, attack budget, or both. In the next budget scenario, we fix all defense variables that are not selected as a part of the solution of the previous problem instance to zero. Thus, the only defense variables that can vary in the next problem instance budget scenario are those defense variables associated with the defenses that were chosen in the previous problem instance. The optimal solution to the new problem instance is determined, and the resulting defenses from that solution will be a subset of the defenses chosen from the starting problem instance. Nesting defenses under decreasing budget scenarios can be called “fixing defenses to zero” since defenses not chosen in larger budget scenarios cannot be chosen in smaller budget scenarios. Equation (3.1d) summarizes the fixing of defenses when a nesting strategy decreases budgets.

$$\hat{W}_{ijd}^{\omega+1} = 0 \quad \forall W_{ijd}^{\omega} : W_{ijd}^{\omega} = 0. \quad (3.1d)$$

C. NESTED DEFENSE HEURISTIC APPROACH

A straightforward approach to constructing a nested defense to a parameterized DAD problem instance is to use the fixed variable approach. We define this straightforward approach to constructing nested defenses as the “nested defense myopic greedy heuristic.” This process of obtaining nested defenses can be defined as a heuristic approach because there is no way of knowing if the nested defenses obtained by this process are the best nested defenses that the network could have. Furthermore, this process can be defined as a greedy heuristic because choosing the next defense variable to nest in the overall defense involves making the best possible improvement to the DAD problem instance with the inclusion of the nesting constraint (Rardin, 1998, p. 699). The process can also be characterized as myopic because it focuses on only one problem instance at a time, and it does not consider all possible budget scenarios at once. We present pseudo-code of a simple algorithm to implement the nested defense myopic greedy heuristic.

Nested Defense Myopic Greedy Heuristic Algorithm

Inputs: DAD problem instance network with a parameterized budget

Outputs: Nested defenses and objective function values for all budget scenarios

Algorithm:

Current budget scenario = starting point problem instance

Nested defense list = empty

Solve DAD problem optimally for starting point budget scenario

While unsolved parameterized budget scenarios exist:

 Append defenses from previous instance solution to nested defenses list

 Include nested defenses as a constraint to next DAD problem instance

 Increment budget scenario by varying parameterized budget by one unit

 Solve DAD problem instance for budget scenario

Step two of the algorithm features a choice of how to start the nesting algorithm. The starting point of a heuristic nesting algorithm is defined as a DAD problem instance for a specific defense budget and attack budget. At the starting point, the DAD problem instance is solved without any nesting requirements. From the starting point, new problem instances are created by either changing the defense budget, the attack budget, or both budgets. The solution of those new problem instances will incorporate the defenses found in the prior problem instance in order for the system to have a nested defense.

Observation 3–1: A problem instance that is chosen as the starting point for a heuristic nesting algorithm has an optimal solution equal to the nested solution.

The choice of the starting point problem instance of the heuristic nesting algorithm can impact the magnitude of the difference between a nested solution and an optimal solution for any specific instance of defenses and attacks. For example, a DAD problem instance with an unknown defense budget could have at least three different starting points. One starting point could be a minimum defense budget and the nesting would increase the defense budget for each successive instance. A second starting point begins at a maximum defense budget and nesting would decrease the budget for each new instance. A third starting point is an intermediate defense budget. Implementation of an intermediate budget starting point would begin with the optimal defense at that budget level and increment the budget until the maximum is reached and then return to the optimal intermediate defense and decrease budgets until the minimum is reached.

As the defense and/or attack budget varies, the potential exists for a difference to grow between the nested defense solution and the optimal solution for different budget scenarios. The choice of the starting point instance has different implications for how the nested solution compares to the optimal solution over all possible defense and attack budgets. For example, consider the DAD problem on the test network where the defense budget is unknown, but the attack budget is fixed at three units. Table 4 summarizes the objective function values for nesting defenses from starting points of zero defense budget, maximum defense budget, and midrange defense budget. The optimal solution for each number of defenses is shaded green. Table 4 shows that the starting point instance for nesting defenses can result in different objective function values.

Table 4. Effect of Defense Budget Starting Point on Nested Defense Solution

Objective Function Values (Attack Budget = 3)	Defense Budget					
	0	1	2	3	4	5
Optimal Solution (Not Nested)	87	80	64	63	47	41
Nested Defense: Start at minimum defense budget	87	80	67	64	49	43
Nested Defense: Start at maximum defense budget	87	87	70	65	47	41
Nested Defense: Start at midrange defense budget	87	87	64	63	55	47

Similarly, different starting points can also result in different nested defenses chosen for a particular defense and attack budget scenario. Table 5 shows the edges chosen as nested defenses for each different nesting start point and compares them with the optimal solution, which is shaded green. Any column in Table 5 shows that the nested defenses chosen depend on the starting point instance.

Table 5. Effect of Defense Budget Starting Point on Nested Defenses Chosen

Defenses Chosen (Attack Budget = 3)	Defense Budget					
	0	1	2	3	4	5
Optimal Solution (Not Nested)	None	(2, 7)	(5,10)	(5, 10) (10,11)	(3, 8) (10, 11) (10, 13)	(3, 8) (9, 13) (10, 11) (10, 13)
Nested Defense: Start at minimum defense budget	None	(10, 13)	(10, 13) (7, 8)	(10, 13) (7, 8) (10,11)	(10, 13) (7, 8) (10, 11) (2, 7)	(10, 13) (7, 8) (10, 11) (2, 7) (9, 13)
Nested Defense: Start at maximum defense budget	None	None	(3, 8)	(3, 8) (10, 13)	(3, 8) (10, 11) (10, 13)	(3, 8) (9, 13) (10, 11) (10, 13)
Nested Defense: Start at midrange defense budget	None	None	(5,10)	(5, 10) (10, 11)	(5, 10) (10, 11) (7, 8)	(5, 10) (10, 11) (7, 8) (10, 13)

1. Heuristic Starting Point Assumption

We assume a nested defense has a starting point scenario that can be computed by finding the optimal solution to a DAD problem instance with a known defense and attack budget. Heuristic nested defense strategies assume that the first budget scenario will have the nested defense equal to the optimal defense. For notation, we will use a star to signify the optimal solution to a DAD problem instance. Z^* refers to the optimal objective function value and W^* refers to the optimal defenses chosen for a DAD problem instance of a specific defense and attack budget. Equation (3.2) states the myopic greedy heuristic starting point scenario assumption.

Nested Defense Heuristic Starting Point Scenario Assumption

$$Z_{ijd}^{''\omega 1''} = \left(Z_{ijd}^{''\omega 1''} \right)^* . \quad (3.2)$$

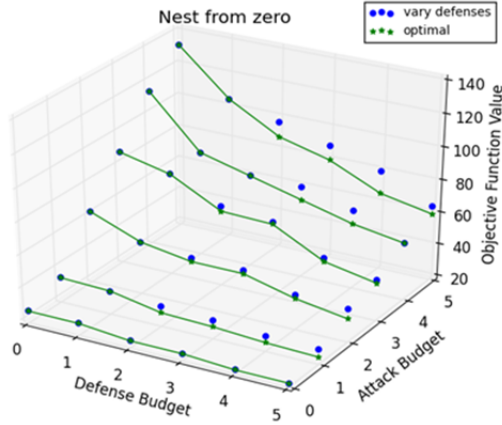
2. Parameterized Defense Budget and Known Attack Budget

When only the defense budget is subjected to parametric analysis, we assume that the attack budget is known. This situation occurs when the defender does not know the defense budget, but must decide on the defenses for the system. The defense budget is revealed after defense decisions are made. The defender must make a prioritized list of the best defenses for the system. The defender chooses a starting point budget scenario for heuristic nesting of prioritized defenses. The starting point instance could be minimum defense budget, maximum defense budget, or somewhere in between.

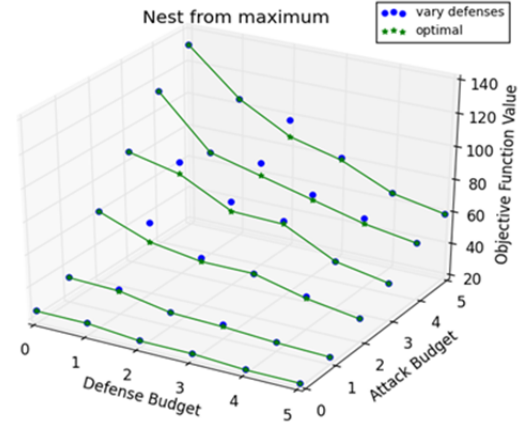
Figure 21 shows how the objective function value of heuristic nested defenses compares with the optimal objective function values for the test network. Figure 21 gives the ability to visualize the effect of nested defenses in comparison to optimal defenses. Problem instance budget scenarios are displayed on two axes, and objective function value is the third axis. The corresponding objective function values are of the nested defense and optimal defense are graphed as pairs of different points. The distance between each pair of points gives the analyst a way to show a decision maker that nesting defenses appears different than the optimal defense in terms of overall system cost.

In Figure 21, the attack budget is held constant while the defense budget is allowed to vary from three different starting points. The first graph starts at zero defense budget and increases the defense budget one unit at a time. The second graph starts at the maximum defense budget of five units and decreases the defense budget by one unit. The third graph starts with a midrange defense budget of three units and increments the defense budget in both directions. The green stars represent the optimal (non-nested) objective function values. The optimal values are connected by line segments to enhance the three dimensional effect of the graph. The blue dots represent the nested solution objective function values. The distance between the green stars and the blue dots represent the cost difference between nested and optimal defenses.

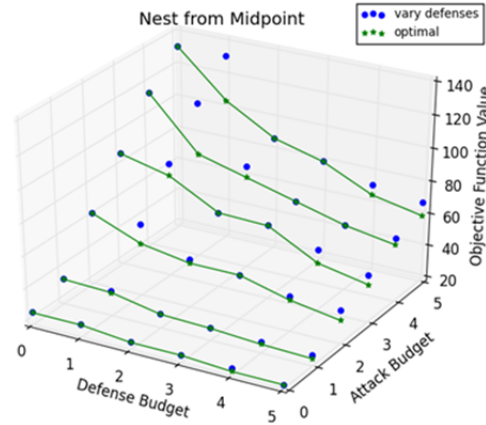
Nested Solutions By Varying Defense



Nested Solutions By Varying Defense



Nested Solutions By Varying Defense



- (a) Starting instance of minimum budget (b) Starting instance of maximum budget
(c) Starting instance of midrange budget

Figure 21. Nesting with Parameterized Defense Budgets at Various Start Points

In the first graph of Figure 21, nesting of defenses begins at zero defense budget. To visualize the effect of increasing defense budgets in the first graph, the reader can follow how the blue dots deviate from the green stars from left to right for any known attack budget. We can see in the first graph that the nested defense solution begins to diverge from the optimal solution when the defense budget is increased to two for all values of attack budgets. The biggest deviations between optimal and nested solutions appear to occur when the attack budget is fixed at larger values and the defense budget is at a midrange value. The deviations from optimality can be explained because the addition of defenses by one budget unit only allows for the protection of existing edges.

Adding a new edge to the network costs two units, and changing the budget by one unit at a time means that adding a new edge is unaffordable. The new edge is unaffordable because only one new unit of defense budget is ever available, and a new edge would require two new units of defense budget. Therefore, a new edge can never be considered as a defense when defense budgets are only increased by one unit at a time.

In the second graph of Figure 21, nesting of defenses begins at the maximum defense budget of five units. To visualize the effect of decreasing defense budgets in the second graph, the reader can follow how the blue dots deviate from the green stars from right to left for any fixed attack budget. The largest deviations between the nested value and the optimal value appear to occur at defense budget of two. This deviation can be explained because the optimal solution at two defenses will choose to add a new edge as a defense, which has a cost of two units. But, the nested defense that is found with a starting point of five units does not have any new edges as defenses, but rather only protected existing edges. Recall that protecting an existing edge only costs one unit. So, as defenses are removed one by one as the defense budget decreases, the nested solution will deviate from the optimal solution because the new edges are not available to be utilized as defense.

In the third graph of Figure 21, nesting of defenses begins at the intermediate defense budget of three units. To visualize the effect of both increasing and decreasing defense budgets in the third graph, the reader can follow how the blue dots deviate from the green stars by starting at the middle of the graph and moving both from right to left and left to right for any known attack budget. In this graph we see the largest deviations when the defense budget is one for most of the fixed attack values. Again, the large deviation can be explained because of how defenses are priced. In most of these cases a new edge defense is present as the only defense when the defense budget is two units. When the defense budget is reduced to one, the new edge defense is no longer affordable, and the system is left defenseless against one attack. The defenseless system against one attack explains the higher nested solution values.

In each of the graphs of Figure 21, we observe that deviations from optimality do not monotonically grow larger as the as the budget scenarios move away from the anchor

point. We believe the largest deviations in all nesting strategies occur when the optimal defense can employ a new edge as a defense, but the nested defense approach cannot. Since the nested defense cannot afford to add a new edge as the defense budget is varied by one unit, the opportunity for large deviations between the solutions occurs.

3. Known Defense Budget and Parameterized Attack Budget

We use the term *persistent defenses* to describe a nested defense DAD problem instance where the defense budget is known in advance, but the attack budget is subject to parametric analysis. In this type of analysis, the defender has to choose a defense to withstand an unknown number of attacks. For the heuristic approach, the defender must choose a starting point instance for the nesting of defenses. The starting point instance can either be a minimum, maximum, or intermediate attack budget.

The concept of nested defenses with a known defense budget and a parameterized attack budget is different than the type of nesting in the previous section. Here, the defenses chosen at the starting point instance do not change over the parameterized attack budget instances because the defense budget is fixed. The nesting requirement does not allow any defense to be swapped out for a different defense as attack budget scenarios change. This fundamental difference in the nature of what the term nested defense means when the defense budget is fixed leads us to introduce the persistent defense terminology in order to avoid confusion.

To understand the effect of persistent defenses, consider the test network and a nested defense requirement for an unknown number of attacks when the defense budget is known to be four units. Tables 6 and 7 show the objective function values and the defenses chosen for a persistent defense from starting points of zero attack budget, maximum attack budget, and midrange attack budget. The non-nested optimal solutions are shaded in green to provide a point of comparison in Tables 6 and 7.

Table 6. Effect of Attack Budget Starting Point on Persistent Defenses

Objective Function Values (Defense Budget = 4)	Attack Budget					
	0	1	2	3	4	5
Optimal Solution (Not Nested)	20	23	37	47	58	65
Persistent Defense: Start at minimum attack budget	20	23	37	62	79	80
Persistent Defense: Start at maximum attack budget	25	33	47	49	64	65
Persistent Defense: Start at midrange attack budget	23	27	45	47	62	71

Table 6 shows that the objective function value for the persistent defenses matches the optimal non-nested value when the number of attacks is equal to the starting point. However, persistent defenses may become suboptimal once the attack budget changes to a level different from the start point. Table 7 provides the detail to show that “persistent” defenses do not change much when the defense budget is a constant. From the starting point of zero attack budget, the same defenses are chosen for any level of attack budget. The same effect is seen with the starting point of attack budget three units.

Table 7. Effect of Attack Budget Starting Point on Persistent Defenses

Defenses Chosen (Def. Budget=4)	Attack Budget					
	0	1	2	3	4	5
Optimal (Not Nested)	(3,8) (5,10)	(3,8) (5,10)	(3,8) (5,10)	(3,8) (10,11) (10, 13)	(1,5) (5,10) (10,11)	(2,7) (7,8) (10,11) (10,13)
Nest start point: Minimum Attack Budget	(3,8) (5,10)	(3,8) (5,10)	(3,8) (5,10)	(3,8) (5,10)	(3,8) (5,10)	(3,8) (5,10)
Nest start point: Maximum Attack Budget	None	None	(2,7) (7,8)	(2,7) (7,8) (10,11) (10,13)	(2,7) (7,8) (10,11) (10,13)	(2,7) (7,8) (10,11) (10,13)
Nest start point: Midrange Attack Budget	(3,8) (10,11) (10, 13)	(3,8) (10,11) (10, 13)	(3,8) (10,11) (10, 13)	(3,8) (10,11) (10, 13)	(3,8) (10,11) (10, 13)	(3,8) (10,11) (10, 13)

There is an anomaly in Table 7 when the starting point is maximum budget because all of the defenses are not retained across the parametrization of attack budgets. For attack budgets of zero, one or two units, all of the defenses chosen at the starting

point are not selected. A simple explanation for this anomaly is that the defenses chosen in Table 7 produce an equivalent objective function value as the attack budget is decremented by one unit at a time. In the starting instance of an attack budget of five units, only existing edges are protected as defenses. There are instances where not protecting all of these existing edges produces an equivalent objective function value. To see this more clearly, consider the instance of defense budget four and attack budget two, depicted in Figure 22. If two existing edges are defended, the objective function value is 47. If four existing edges are defended, the objective function value is also 47. The worst-case attacks are different for these equivalent defense choices, but the resulting objective function values are the same. It is not obvious in Figure 22 that these two defenses are equivalent, since no nodes are isolated from flow in the system on the left, but nodes 1, 5 and 9 are isolated in the system on the right. These two systems are equivalent because the transport costs for the system on the left offset the penalties for unmet demand at nodes for the system on the right.

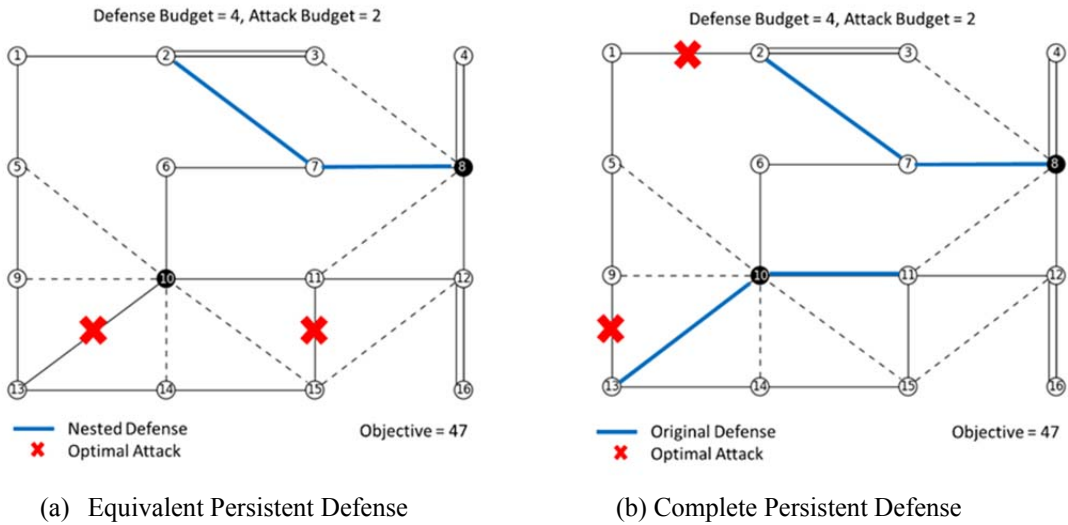
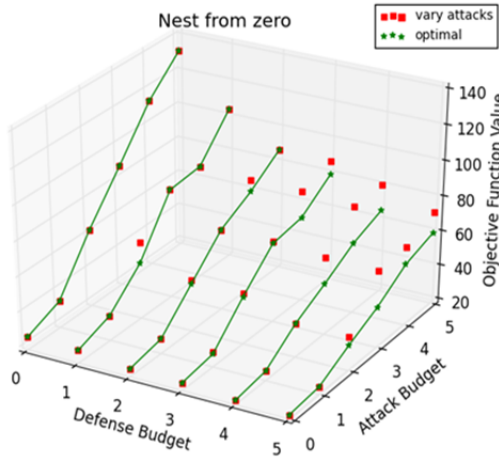


Figure 22. Equivalent Persistent Defenses on the Test Network

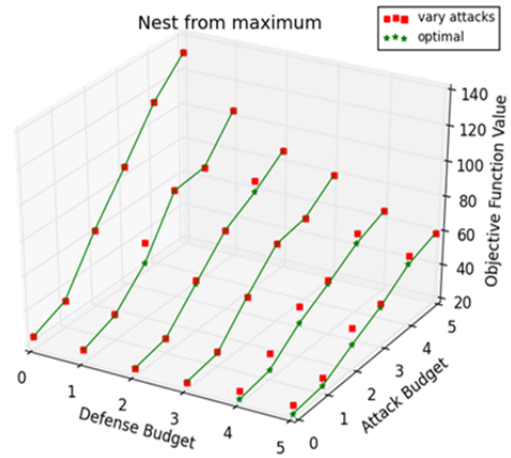
The other interesting effect seen in Tables 6 and 7 is the persistent defense is also optimal when there are zero, one or two attacks with a starting point instance of minimum attack budget. The same defense is optimal for these different attack budgets.

The next series of graphs show how the objective function value of persistent defenses compares with the optimal objective function values. In Figure 23, the defense budget is held constant while the attack budget is allowed to vary from three different starting points. The first graph starts at zero attacks and increases the attack budget one unit at a time. The second graph starts at five attacks and decreases the attack budget by one unit. The third graph starts with three attacks and increments the attack budget in both directions. The green stars represent the optimal (non-nested) objective function values. The optimal values are connected by line segments to enhance the three dimensional effect of the graph. The red squares represent the objective function values for the system with persistent defenses. The distance between green stars and red squares represent the cost difference between the persistent defense and the optimal defense.

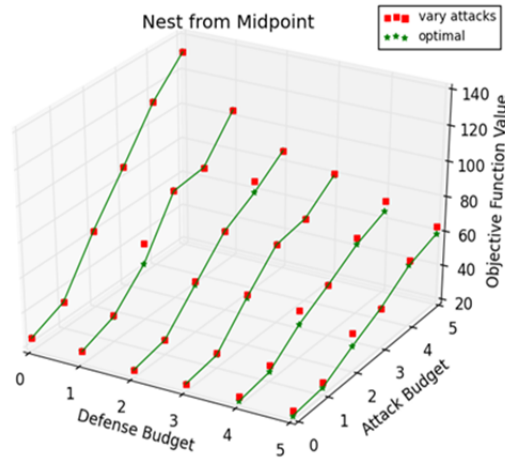
Persistent Defenses By Varying Attack



Persistent Defenses By Varying Attack



Persistent Defenses By Varying Attack



- (a) Starting instance of minimum budget (b) Starting instance of maximum budget
(c) Starting instance of midrange budget

Figure 23. Persistent Defenses with Parameterized Attack Budgets

In the first graph of Figure 23, persistence of defenses begins at an attack budget of zero. To visualize the effect of increasing attack budgets in the first graph, the reader can follow how the red squares deviate from the green stars from the front to the back for any fixed defense budget. We can see in the first graph that the nested defense solution begins to diverge from the optimal solution when the defense budget is increased to two units. The biggest deviations between optimal and persistent defense appear to occur when the defense budget and attack budget are large values. The large deviations from optimality can be explained at these instances because the persistent defenses are optimal

for zero attacks, and they remain the same for all attack budgets. These defenses are less likely to be optimal when more attacks are applied to the network, and the objective function suffers more and more as the number of attacks against these defenses increases.

In the second graph of Figure 23, persistence of defenses begins at the maximum attack budget of five units. To visualize the effect of decreasing attack budgets in the second graph, the reader can follow how the red squares deviate from the green stars from the back of the graph to the front of the graph for any fixed defense budget. We can see in the second graph that the magnitude of the deviations from optimality appears to be smaller overall than the first graph. The biggest consistent deviations between optimal and persistent defense appear to occur for defense budgets of four and five. An explanation for this behavior is that the persistent defense is optimal only when there are five attacks, and less attacks on the system with the same defenses result in larger costs than optimal defenses.

In the third graph of Figure 23, persistence of defenses begins at a midrange attack budget of three units. To visualize the effect of increasing and decreasing attack budgets in the third graph, the reader can follow how the red squares deviate from the green stars from the middle of the graph to the back of the graph and again from the middle of the graph to the front of the graph for any fixed defense budget. We can see in the second graph that the magnitude of the deviations from optimality appears to be smaller overall than the first graph. The biggest consistent deviations between optimal and persistent defense appear to occur for attack budgets of two and four units for almost all fixed values of defense budgets. All of the deviations appear to be fairly small without an obvious explanation.

4. Parameterized Defense and Attack Budgets

Nested defenses for the DAD problem are more complicated when both the defense and attack budgets are unknown. We believe this situation is the most difficult and realistic type of scenario that an analyst can face when a prioritized list of nested defenses is required. First, we show that the analyst should not use the techniques of the

previous two subsections because the results will be inconsistent. Second, we propose a strategy to parameterize both the defense and attack budget to achieve nested defenses.

a. Use of Previously Seen Nesting Methods is Inconsistent

When both the defense and attack budgets are unknown, the defender should not use one of the previous nesting strategies. Nesting strategies which only parametrize one budget may lead to inconsistent nested solutions when both attack and defense budgets are uncertain. For example, the analyst could pretend that he or she knows the defense budget and then vary the attack budget. Then the analyst could repeat the process for each possible defense budget level to obtain a set of nested defenses. Alternatively, the analyst could pretend that the attack budget is known and then vary the defense budget. Next, the analyst would repeat the process for every possible attack budget and obtain a set of persistent defenses. Both of these approaches are not guaranteed to produce a similar set of solutions that are completely nested across all budget scenarios. Both of these nesting methods guarantee a nested solution along one axis (defense or attack budget), but it does not guarantee that a solution set will be nested on the other axis. An example can be seen in the test network.

In the test network we see that the nested solution for a defense budget of two units and an attack budget of three units can produce two different nested defense solutions with different objective function values if the nesting strategy is changed, even when similar nest starting points are used. The nested solution obtained by varying the defense budget and holding the attack budget constant yields an objective function value of 67 and a defense plan to defend two existing edges. But, when the attack budget is varied and the defense budget is held constant, the nested solution yields an objective function value of 64 and a defense plan to build one new edge. Tables 8 and 9 summarize the different results that are obtained when only one budget is parameterized. In Table 9, note that only one defense is chosen with the defense budget of two units because the cost of adding edge (5, 10) in the test network is two units.

Table 8. Nested Defense Obtained by Varying Defense Budget

Defense and Attack Budget Unknown	Defense Budget		
	0	1	2
Attack Budget = 3 Nest Start: Minimum Defense Budget	Objective = 87 Defend: None	Objective = 80 Defend: (10, 13)	Objective = 67 Defend: (10, 13), (7, 8)

Table 9. Nested Defense Obtained by Varying Attack Budget

Defense and Attack Budget Unknown	Attack Budget			
	0	1	2	3
Def. Budget = 2 Nest Start: Min. Attack Budget	Objective = 22 Defend: (5, 10)	Objective = 26 Defend: (5, 10)	Objective = 47 Defend: (5, 10)	Objective = 64 Defend: (5, 10)

In Tables 8 and 9, the scenarios shaded in orange both have the same defense budget of two and attack budget of three, but the objective function values and defenses are different. When both the defense budget and attack budget are uncertain, the analyst cannot arbitrarily choose to hold one budget constant and incrementally change the other budget. The arbitrary choice of which budget to hold fixed and which to vary affects both the objective function value and the defense plan obtained.

b. V Nesting Method

A different method is needed to produce nested solutions to DAD problem instances under uncertain defense and attack budgets. This method must be able to produce defenses that are nested and also account for the uncertainty in both the defense and attack budgets. We have developed a nesting algorithm called *V nesting* that can account for both types of budget uncertainty. Instead of nesting by either holding the defense or attack budget constant, this method increases both budgets simultaneously. This algorithm chooses a series of successive start points called *anchor points* and builds nested defenses along each uncertain budget axis. The base of the V is the anchor point instance, and each arms of the V is the nested instance resulting from varying one of the budget parameters. This method ensures that each successive anchor point has nested

defenses from the previous anchor point as well. Since each axis builds nested defenses from its respective anchor point, the group of problem instances that are defined by a particular anchor point resembles a “V” shape when plotted on a graph with the defense budget and attack budget as the axes. We observe that V nesting is not the only possible strategy for nesting defenses when both defense and attack budget are subject to parameterization.

The V nesting algorithm works in the following fashion. Assume that both the defense budget and attack budget are not known to the analyst. The analyst chooses a starting point scenario for nesting, such as minimum defense and attack budgets. The starting point scenario serves as an anchor instance to nest defenses for similar instances. We define a *similar instance* to be a DAD problem instance that has either the same defense budget or attack budget to the anchor problem instance. From the starting point solution, nested defenses to similar problem instances are obtained in two steps. The first step is to generate new problem instances by holding the attack budget constant, and increase the defense budget incrementally by one unit to the maximum parameter value. Each new problem instance is solved with the nested defense requirement. The second step is to generate more new problem instances by holding the defense budget at minimum and increasing the attack budget by one unit until the maximum attack budget is reached. No new defenses are added in the second step, but the new problem instances will have a defense that is the same as the starting point defense. Thus, all similar problem instances that either increase the defense budget or the attack budget, but not both, must have a nested defense in comparison to the anchor instance. Next, another anchor problem instance is created by taking the last known anchor point and increasing the defense and attack budgets by one unit and obtaining a nested defense based on the previous anchor. The process repeats at the new anchor point. Anchor points are successively created until all defense budget and attack budget combinations have been evaluated. The pseudo-code of the V-nesting algorithm summarizes how the algorithm forms a nested defense.

“V” nesting algorithm (increasing defense & attack budget)

Inputs:

DAD minimum cost flow network with parameterized defense & attack budgets

Defense level = minimum defense budget (zero for test network)

Attack level = minimum attack budget (zero for test network)

Outputs: Objective function values and nested defenses for all budget scenarios

Algorithm:

While defense level < maximum budget **OR** attack level < maximum budget:

 Solve “Anchor” DAD problem instance

 Append defenses obtained to nested defense list

While defense level < maximum defense budget:

 Increment defense level by one unit

 Solve “similar” DAD problem instance

 Append defenses obtained to nested defense list

 Reset defense level = anchor defense budget

 Reset nested defense list to anchor nest defense list

While attack level < maximum attack budget:

 Increment attack level by one unit

 Solve “similar” DAD problem instance

 Append defenses obtained to nested defense list

 Reset attack level = anchor defense budget

 Reset nested defense list to anchor nest defense list

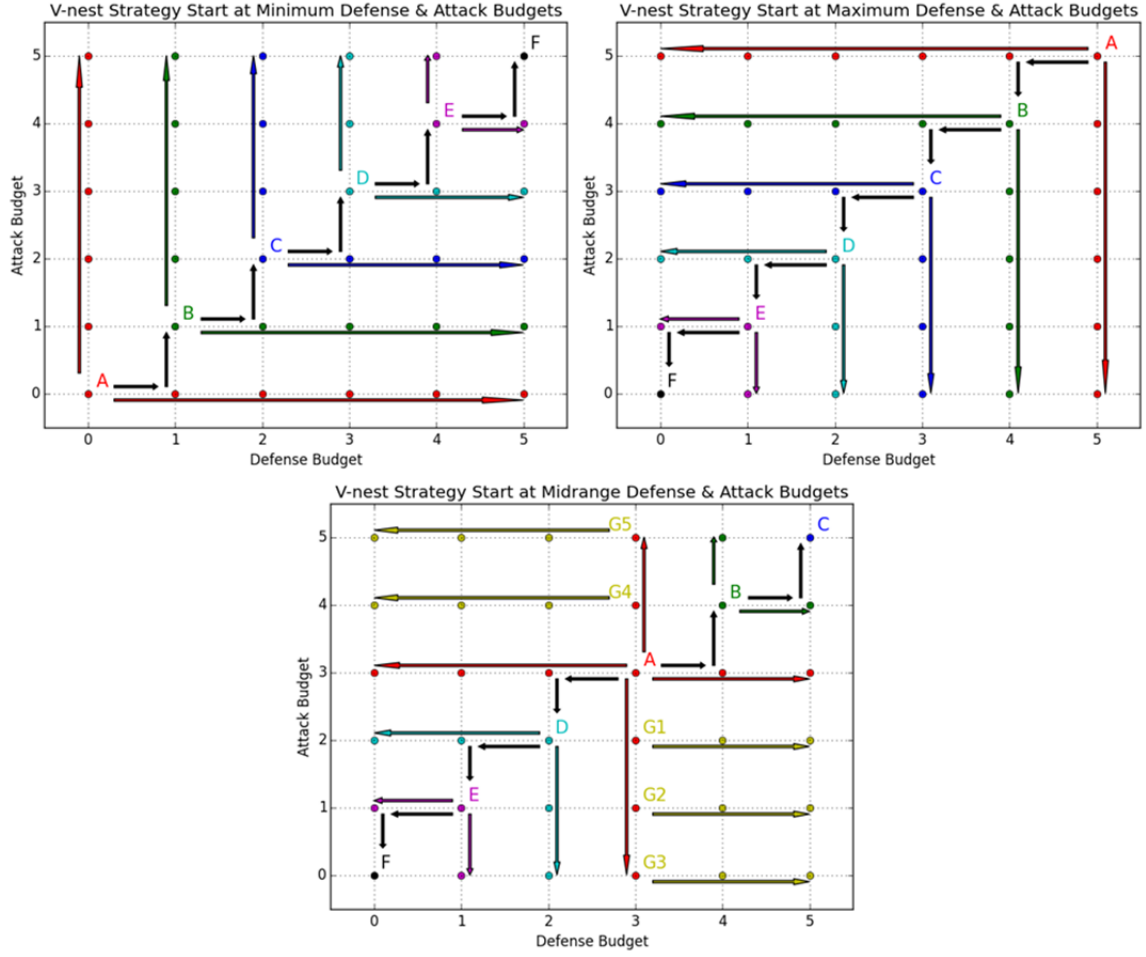
 Create new “anchor” DAD problem instance:

 Reset nested defense list = nested defense list for last anchor point

 Increment attack level by one unit

 Increment defense level by one unit

Figure 24 illustrates how successive problem instances are created for different starting point instances of the V nesting algorithm. The axes of each plot are defense budget and attack budget levels. Each point on the grid represents a DAD problem instance for a specified defense budget and attack budget. The points with assigned letters represent the anchor points for nesting of defenses. All points of the same color have a nested defense. Also, the assigned letter points have nested solutions from each other. The colored arrows describe how nested defenses successively build from each anchor point. The black arrows show how each anchor point is nested from a previous anchor.



(a) Starting instance of minimum budget (b) Starting instance of maximum budget
(c) Starting instance of midrange budget

Figure 24. Nesting Strategies for Parameterized Defense and Attack Budgets

The first plot of Figure 24 helps to explain the V-nesting strategy from a starting point of minimum defense and attack budgets. First, we start at point *A*, which represents a DAD problem instance of zero defense and attack budget. The optimal solution to this instance is the starting point for the nesting of all defenses. Next, the horizontal red arrow depicts that nested solutions for problem instances of zero attacks and various defenses are determined by increasing defense budget by one unit, with each solution nested by the previous solution. The vertical red line depicts that nested solutions for zero defenses are obtained by increasing the number of attacks by one unit, and holding the defense budget at zero. Obviously, there are no defenses chosen for any of these instances, since there is

no defense budget available. In order to obtain a nested solution at point B , we follow the nesting black line from the instance at point A to the instance of one defense and zero attacks and then increase the defense budget by one unit from that instance to obtain the new nested defense instance at point B . The process repeats itself at point B to form all of the green problem instances that have nested solutions that are nested from the instance at point B . Nested defenses for problem instances derived from points C , D , E and F are determined in the same fashion. Each problem instance of an equal number of defenses and attacks then has a nested solution set. Each of these instances becomes a base for a V shape of problem instances that either vary the attack or defense budgets. Nested solutions are built from these base instance solutions to complete the nesting strategy.

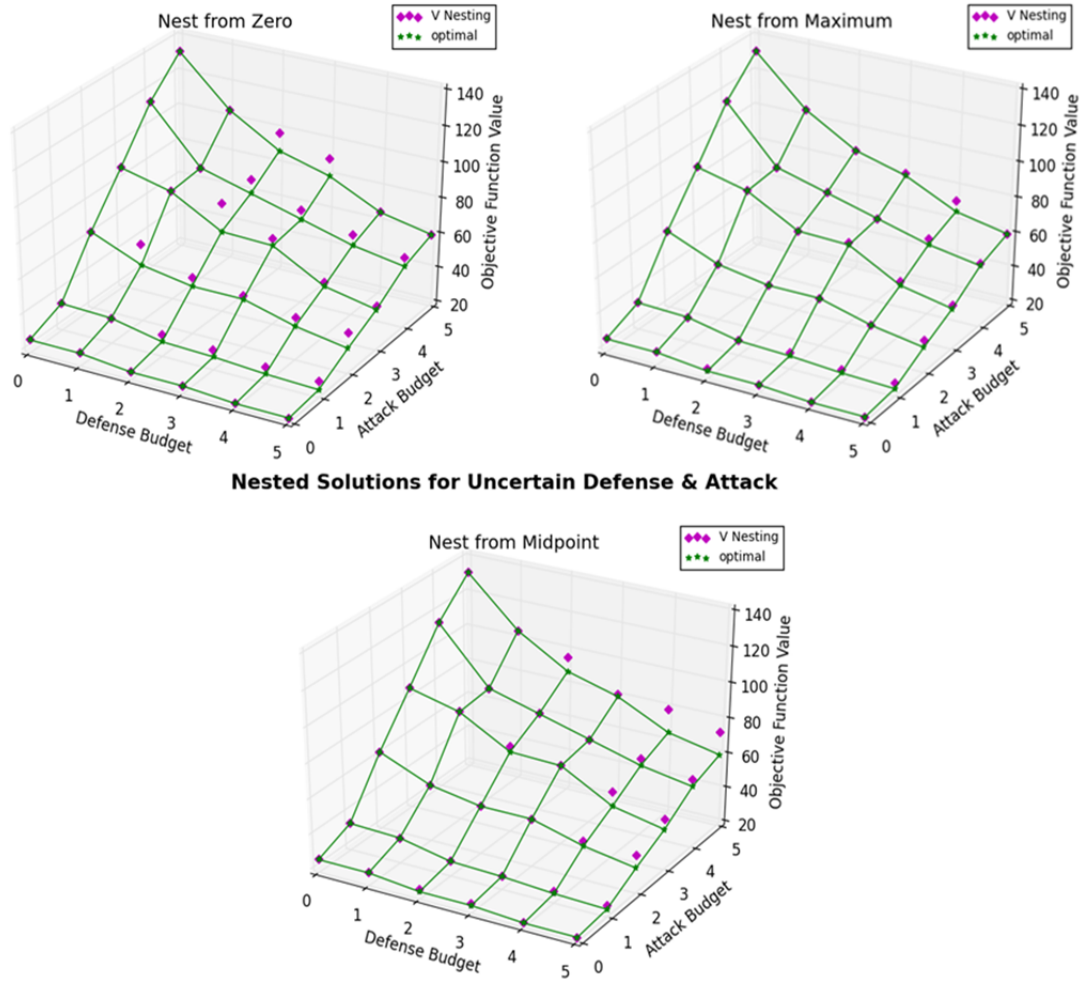
The V-nesting strategy for a starting point instance of maximum defense and attack budgets is shown in the second plot of Figure 24. This nesting start point requires a few modifications to the previously discussed algorithm. The first anchor point would be initialized to maximum defense and attack budgets. The initial nested defenses list contains the defenses utilized in the optimal solution to the anchor point budget scenario. As successive DAD problem instances are solved, the nested defense list has items removed instead of appended. The starting point is at problem instance A on the second plot. The development of nested defenses from point A is derived in a similar fashion. In order to nest from anchor point A to anchor point B , the nested defense for one less defense budget unit is also nested as the attack budget is reduced by one unit. The black arrows show how the nesting is performed. The process is repeated for anchor points B through F to complete the V-nesting strategy.

The V-nesting strategy is slightly more complicated when starting from a midrange defense and attack budget. The third plot of Figure 24 shows how the nesting strategy grows from a midrange primary anchor point. The first anchor point A is solved optimally. The increasing budget nesting strategy is used to determine all problem instances from anchor points A through C . The decreasing budget nesting strategy is used for problem instances that can be defined in terms of anchor points A , D , E , and F . However, the problem instances represented by the yellow dots in the third plot are not defined in terms of either of these nesting strategies. In order to have nested defenses at

these instances, we define additional anchor points from previously defined problem instances that are nested from the first anchor point A . These additional anchor point problem instances are labeled $G1$ through $G5$. Each of these five anchor point instances has a nested defense from the first anchor point A . From each of the G anchor points the defense budget is increased or decreased, as necessary, to define the remaining problem instances. All instances have a defense that is nested in terms of the starting point problem instance of anchor point A .

The next three graphs show how the objective function value of V-nested defenses compares with the optimal objective function values. In Figure 25, the effect on the objective function value of nesting defenses with a parameterized defense and attack budgets is compared with optimal objective function values. The defense and attack budgets are allowed to vary from three different starting points. The first graph starts at zero defense and attack budgets. The second graph starts at defense and attack budgets at a maximum of five units. The third graph starts at a midrange defense and attack budget of three units each. The green stars represent the optimal (non-nested) objective function values. The optimal values are connected by line segments to enhance the three dimensional effect of the graph. The magenta diamonds represent the nested defense solution objective function values. The vertical distances between green stars and magenta diamonds represent the objective function value difference between the nested defense and the optimal defense.

Nested Solutions for Uncertain Defense & Attack Nested Solutions for Uncertain Defense & Attack



(a) Starting instance of minimum budget (b) Starting instance of maximum budget
(c) Starting instance of midrange budget

Figure 25. Nesting with Parameterized Defense and Attack Budgets

In the first graph of Figure 25, nesting of defenses begins at a defense and attack budget of zero. To visualize the effect of increasing defense and attack budgets in the first graph, the reader can follow how the magenta diamonds deviate from the green stars from the bottom left corner of the graph to the top right corner of the graph. We can see in the first graph that the nested defense solution begins to diverge from the optimal solution when the attack budget increases, but the bigger deviations occur when the defense budget is increased to two. This can be explained because at a defense budget of two units, the optimal solution can add new edges as defense to the system. But, the V-

nesting strategy cannot use these defenses because it only increments defenses by one unit at a time. Since only one new unit of defense budget is ever available to the algorithm, adding a new edge is unaffordable and never selected.

In the second graph of Figure 25, nesting of defenses begins at the maximum defense budget and attack budget of five units. To visualize the effect of decreasing defense and attack budgets in the second graph, the reader can follow how the magenta diamonds deviate from the green stars from the top right corner to the bottom left corner of the graph. The second graph shows that in general terms, the V-nesting approach is a very close to the optimal solution when beginning from a starting point of maximum budgets. However, we believe that a unique aspect of the problem instance allows this method to have the smallest deviation. The optimal solution of the maximum defense and attack budget problem instance involves defending five existing edges, and defending an existing edge costs one unit. Thus, removing one unit of budget at a time will result in defending a corresponding amount of existing edges to the parameterized defense budget. The added edges costing two defense units do not appear in the anchor scenario for this problem instance, so large deviations from optimality do not occur.

In the third graph of Figure 25, nesting of defenses begins at a defense and attack budget of three units. To visualize the effect of changing defense and attack budgets in the second graph, the reader can follow how the magenta diamonds deviate from the green stars from the center to the bottom left corner of the graph and again from the center to the top right corner of the graph. We can see in the third graph that deviations between the nested and optimal solutions are very small from the starting point to the minimum budget levels. However, deviations between the nested and optimal solutions do develop from the middle and are fairly substantial when the defense budget or the attack budget are at their maximum values of five units.

D. NESTED DEFENSE ALGORITHM STEP SIZE

Next, we change an assumption on how a nesting of defenses operates. Previously we assumed that nesting of defenses required an incrementing of the parameterized budgets by one unit. We show that this definition of nesting can be problematic. We will

broaden the definition of nesting so that the incremental change of budgets does not have to be by one unit. The size of the incremental change of budget can be referred to as the “step size” of the nesting requirement. Define a nesting step size to be a nesting of defense budget by an increment other than one. For example, nesting could occur by every two defense budget units, so that the five best defenses includes the three best defenses, and the three best defenses includes the one best defense. Generally, a nesting requirement of k budget units would mean that the k best defenses would be a subset of the $2k$ best defenses, which would also be a subset of the $3k$ best defenses, and so on.

We want to examine the step size between nested defenses because not all defenses have the same cost in the test network. Defending an existing edge costs one unit of defense budget, but adding a new edge to the network costs two defense budget units. We are never be able to add a new edge to the network if we use a nesting strategy that adds to the defense budget one unit at a time. Since we only have one more unit of budget available, the only option available is to defend another existing edge.

To illustrate why the use of a nested step size of one defense can be problematic, consider an example from the test network using the myopic greedy heuristic algorithm. We will consider nesting strategy of increasing defenses, and use the instance of defense and attack budgets of one unit as a starting point. The optimal solution to the DAD problem instance is to defend the existing edge (10, 13) with an objective function value of 32. In order to obtain the nested defense for the instance of defense budget of two and attack budget of one, the defense of edge (10, 13) must be included in the solution. The best feasible nested defense solution for defense budget of two and attack budget of one is to defend edges (10, 13) and (2, 7), with an objective function value of 30. However, the optimal solution for the defense budget of two and attack budget of one problem instance is to add new edge (5, 10) with an objective function value of 26. The problem that can be seen in the one step nesting of defenses approach is that it would be impossible for the nesting algorithm to ever even consider building new edge (5, 10) because it has a defense cost of 2 units. When the defense budget is increased from one unit to two units, there is actually only one new unit of defense budget that can be used, since the nesting requirement of defending edge (10, 13) requires the consumption of one

unit of defense budget. Therefore, the nesting algorithm can only choose a second defense that costs one unit of defense budget. The only defense options available in the test network that cost one unit are defending existing edges. A nesting algorithm with a step size of one is forced to exclude adding new edges, since this type of defense has an unaffordable cost of two units. Since the step size of one nesting algorithm cannot consider the defense that leads to optimal solution for the defense budget of two and attack budget of one problem instance, it is impossible for it to obtain the optimal solution. The plots of Figures 26 show the nested and optimal solutions on the test network. A more general statement from this example is given in an observation.

Observation 3–2: A nested solution algorithm that has a step size smaller than the cost of any particular defense option cannot ever consider that particular defense option for a feasible solution, except during the anchor point budget scenario.

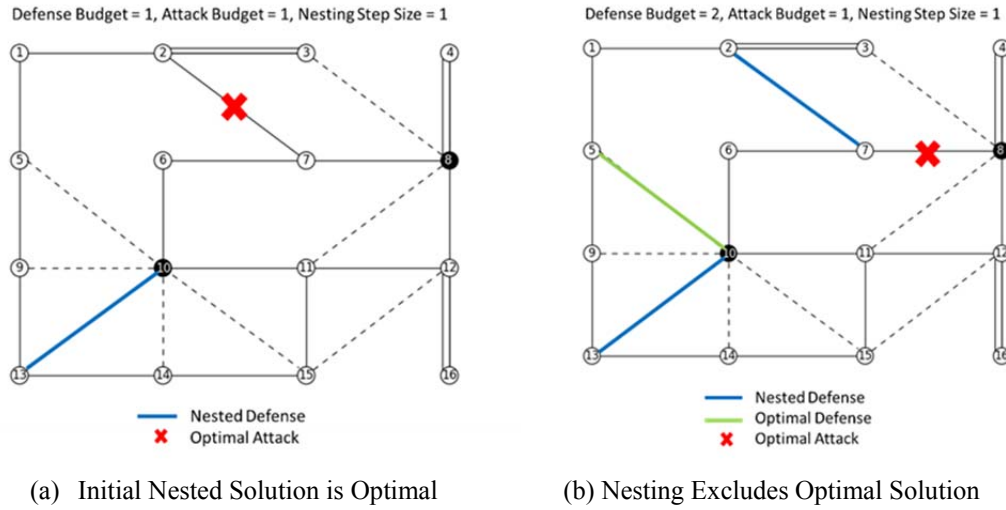


Figure 26. Effect of Budget Step Size on Nested Defense

In order to prevent the situation in the Figure 26, a nesting solution should have a minimum defense budget step size that can allow the choice of any defense option. In the case of the test network, defending existing edges costs one unit and adding new edges costs two units, so the minimum step size should be two units of defense budget. Notice that the minimum step size of the defense budget has nothing to do with the minimum

step size of the attack budget. But, a similar problem could exist if attacking different items in the system had different costs to the attacker. In that case, then the minimum step size in the attack budget should be equal to the cost of the largest attack, so that every attack could be considered.

In Figure 27, the effect on the objective function value of nesting defenses with a parameterized defense budget and a known attack budget is compared with optimal objective function values with a defense budget step size of two units. The number of attacks are held constant while the number of defenses is allowed to vary from two different starting points. The first graph starts at defense budget of one and increases the defense budget two units at a time. The second graph starts at a maximum defense budget of five units and decreases the defense budget by two units. The green stars represent the optimal (non-nested) objective function values. The optimal values are connected by line segments to enhance the three dimensional effect of the graph. The blue dots represent the nested solution objective function values. The vertical distances between the green stars and the blue dots represent the objective function difference between the nested defense and the optimal defense.

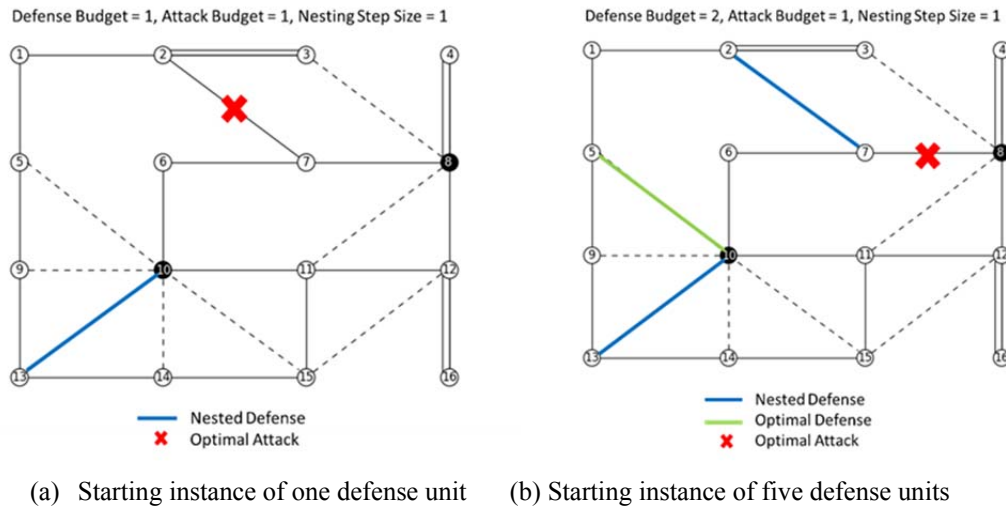


Figure 27. Nested Defenses for a Two Unit Step Size

In the first graph of Figure 27, nesting of defenses begins at a defense budget of one unit. To visualize the effect of increasing defense budgets in the first graph, the reader can follow how the blue dots deviate from the green stars from left to right for any fixed attack budget. We can see in the first graph that the nested defense solution almost perfectly matches the optimal solution as the defense budget is increased from one to three units for every instance of known attack budgets. There is only a small difference between the nested solution and the optimal solution as the defense budget is raised again from three units to five units for most instances of fixed attack budgets. In general, it appears that the use of a larger step size makes the nested defense almost as good as the optimal defense.

In the second graph of Figure 27, nesting of defenses begins at a maximum defense budget of five units. To visualize the effect of decreasing defense budgets in the second graph, the reader can follow how the blue dots deviate from the green stars from right to left for any fixed attack budget. We can see in the second graph that the nested defense is a nearly perfect match to the optimal defense when the defense budget is decreased from five to three units. When the defense budget is lowered again from three units to one unit, the nested defense is a perfect match to the optimal solution in all but one instance. The only deviation between the nested defense and the optimal defense occurs when the known attack budget is two units. Again, it appears that the use of a step size greater than one makes the nested defense almost as good as the optimal defense.

The shortcoming of this approach is that some problem instances are not defined because the defense budget was skipped by the larger step size. It is possible that in a scenario where the defense budget is uncertain, that a defense budget of four is realized, but the nesting analysis only covers defense budgets of one, three and five. The action that should be taken by an analyst in this situation is unclear.

The prioritized list gives the defenses that should be chosen for the next lower defense budget. We recommend the implementation of those defenses in this case. It would be possible to have no extra defenses and leave the remaining portion of the defense budget unused. This course of action is most likely suboptimal since an additional defense may force the attacker to change their plans. The other course of

action would be to use the defenses chosen in the prioritized list for the next lowest defense budget, and then choose one more defense to completely exhaust the known defense budget through another round of analysis.

It is unnecessary to perform an analysis of the effects of step size for persistent defenses when the defense budget is known in advance and the attack budget is unknown on the test network. Recall that in our test network all attacks have an equal cost of one unit. There are no attacks that have costs greater than one, so there is no need to undergo an investigation of larger step sizes of persistent defenses, since all possible attacks can be considered with a step size of one unit of attack budget.

E. NESTED DEFENSE DISTANCE FROM OPTIMALITY MEASUREMENT

When considering how to assess the optimality costs of a nested defense to a DAD problem instance, a methodology to measure the distance between the nested defenses and the optimal defenses is necessary. In our analysis so far, we have seen how to graphically compare nested solutions to optimal solutions for all budget scenarios. Some insight can be gained from looking at plots of nested defenses and optimal defenses for all budget scenarios. But, the graphs presented in the last section do not provide conclusive proof that one nesting strategy is preferable to another.

We desire one simple number as an assessment of the *quality* of the prioritized list of nested defenses. We need to measure a *distance* from optimality for a nested set of defenses to a DAD problem instance. The concept of measuring the cost of nested defenses has been touched upon briefly in the literature. Koc and Morton define the “cost of prioritization” as the “percentage difference of the optimal prioritization solution from the wait-and see solution” (2015, p. 591). The *wait and see solution* refers to the optimal solution without a prioritized list of nested defenses. We examine five different methods to measure the cost of prioritization.

1. Vector Analysis

The nested defense objective function values for parameterized defense and/or attack budget scenarios can be formed into a vector. Similarly, the corresponding optimal

non-nested objective function values for these parameterized budget scenarios can also be formed into a vector. We consider the optimal non-nested solution to the problem as a common point of reference or *zero vector* to compare against any nested defense solution vector. We define the *difference vector* as the vector of differences between the optimal zero vector and the nested solutions vector. The goal of the vector norm analysis is to determine which nesting strategy is the closest to the optimal solution when a choice of nesting strategies are available to an analyst.

The difference vector gives the detail of how far each nested solution is from the optimal solution. All of the parametrized budget scenarios are captured as elements in the difference vector. Vector norms can be computed on the difference vector to give a one number comparison of a nested solution strategy with the optimal solution. Leon (2010, p. 238) defines common vector norms. The 1-norm is the sum of the absolute value of all differences. The 2-norm is the square root of the sum of all squared differences. The infinity norm is the maximum absolute value difference within the vector of differences. These norms are summarized in Figure 28. The nesting strategy that has the smallest valued norms would be the closest to the optimal solution. A decision maker should choose a nesting strategy with the smallest vector norms.

Vector Norm Analysis Definitions:

$b \in B$	Set of all defense budget values; (Note: b_{min} & b_{max} are minimum & maximum values.)
$a \in A$	Set of all attack budget values (Note: a_{min} & a_{max} are minimum & maximum values.)
$Optimal_{ba}$	Optimal objective function value for budgets b & a .
$Nested_{ba}$	Nested objective function value for budgets b & a .
$\delta_{ba} = nested_{ba} - optimal_{ba}$	Difference between solutions for budgets b & a .

$$V = \begin{bmatrix} \delta_{b_{min}, a_{min}} \\ \vdots \\ \delta_{b_{max}, a_{max}} \end{bmatrix}, \quad \|V\|_1 = \sum_{\substack{b \in B, \\ a \in A}} |\delta_{ba}|, \quad \|V\|_2 = \left(\sum_{\substack{b \in B, \\ a \in A}} |\delta_{ba}|^2 \right)^{1/2}, \quad \|V\|_\infty = \max_{\substack{b \in B, \\ a \in A}} |\delta_{ba}|.$$

Figure 28. Definitions of Vector Norms. Adapted from Leon (2010).

Observation 3–3: Each difference between the nested defense value and the optimal value must be a non-negative number.

Proof of Observation 3–3: It is impossible to achieve any feasible solution to a problem instance that is better than the optimal solution.

In order to better understand how the vector norm allows us to assess the performance of a nesting strategy, consider the test network with the myopic greedy heuristic nesting strategy where both the defense budget and the attack budget are unknown. Our solution performs the V-nesting strategy from three different starting point instances. The goal of the vector norm analysis is to determine whether the starting point of minimum, midrange, or maximum budget is a better strategy for V-nesting. Define the vectors V^{MIN} , V^{MID} and V^{MAX} to be the difference vectors associated with starting points of minimum, midrange and maximum defense and attack budget scenarios, respectively. Table 10 shows the associated norms for V^{MIN} , V^{MID} and V^{MAX} on the test network with parameterized defense and attack budgets.

Table 10. Vector Norm Analysis at Different Starting Point Instances

Test Network –Defense and Attack Budget Unknown	V^{MIN}	V^{MID}	V^{MAX}
Vector 1-norm: $\ V\ _1$	118.0	75.0	28.0
Vector 2-norm: $\ V\ _2$	31.3	24.7	9.5
Vector ∞ -norm: $\ V\ _\infty$	16.0	13.0	6.0

In Table 10 we can see that the norms associated with the V^{MAX} vector are always smaller than the norms associated with the V^{MID} and V^{MIN} vectors. Therefore, an analyst should choose the nesting strategy start point of maximum defense and attack budgets because it is closest to the optimal solution.

2. Matrix Analysis

It is also possible to analyze the difference between nested defense solutions and optimal solutions in a matrix we call the *difference matrix*. Rows of the matrix represent the defense budget in a problem instance, in ascending order from minimum to maximum. Columns of the matrix represent the attack budget in a problem instance, also

in ascending order from minimum to maximum. Any particular element in the difference matrix represents the difference between the nested defense objective function value and the optimal objective function value for a specific defense and attack budget.

The use of the difference matrix to calculate a distance between a set of nested defenses and the corresponding optimal defenses can be performed with matrix norms shown in Figure 29. The matrix 1-norm represents the maximum absolute column sum (Leon, 2010, p. 407). The matrix 1-norm corresponds to fixing the number of attacks and calculating the maximum sum of differences across changing defense budgets. The matrix infinity norm represents the maximum absolute row sum (Leon, 2010, p. 407). The matrix infinity norm calculates the maximum sum of differences as the attack budget changes. We can also use the matrix Frobenius norm, which is defined as the square root of the sum of squares of all of the elements in the matrix (Leon, 2010, p. 235).

$$M = \begin{bmatrix} \delta_{b_{\min}, a_{\min}} & \cdots & \delta_{b_{\min}, a_{\max}} \\ \vdots & \ddots & \vdots \\ \delta_{b_{\max}, a_{\min}} & \cdots & \delta_{b_{\max}, a_{\max}} \end{bmatrix},$$

$$\|M\|_1 = \max_{b \in B} \sum_{a \in A} |\delta_{ba}|,$$

$$\|M\|_F = \left(\sum_{\substack{b \in B, \\ a \in A}} |\delta_{ba}|^2 \right)^{1/2},$$

$$\|M\|_\infty = \max_{b \in B} \sum_{b \in B} |\delta_{ba}|.$$

Figure 29. Description of Matrix Norms. Adapted from Leon (2010).

Each of these matrix norms is best suited to a particular type of parametric programming analysis. The matrix 1-norm would be best when the defense budget is subject to parameterization and the attack budget is known. The matrix infinity norm would be best when the attack budget is known and the defense budget is subject to parameterization. Lastly, an analysis of the Frobenius norm would make sense when both the attack and defense budget are unknown.

To understand why the matrix 1-norm is best for unknown defense budgets, suppose the defense budget is parameterized and the attack budget is known to be two

units. In this case, the matrix of all possible defense and attack budget combinations reduces to a simple column vector, as shown in Figure 30.

$$M = \begin{bmatrix} \delta_{b_{\min}, a_{\min}} & \cdots & \delta_{b_{\min}, a_{\max}} \\ \vdots & \ddots & \vdots \\ \delta_{b_{\max}, a_{\min}} & \cdots & \delta_{b_{\max}, a_{\max}} \end{bmatrix} \rightarrow \begin{bmatrix} \delta_{b_{\min}, 2} \\ \vdots \\ \delta_{b_{\max}, 2} \end{bmatrix}$$

Figure 30. Use of the Matrix 1-Norm for Unknown Defense Budgets

Conversely, to see why the matrix infinity-norm is best for the unknown attack budget situation of “persistent defenses,” suppose the defense budget is fixed at two units and the attack budget is parameterized. In this case, the matrix of all possible defense and attack budget combinations reduces to a simple row vector, as shown in Figure 31.

$$M = \begin{bmatrix} \delta_{b_{\min}, a_{\min}} & \cdots & \delta_{b_{\min}, a_{\max}} \\ \vdots & \ddots & \vdots \\ \delta_{b_{\max}, a_{\min}} & \cdots & \delta_{b_{\max}, a_{\max}} \end{bmatrix} \rightarrow [\delta_{2, a_{\min}} \quad \cdots \quad \delta_{2, a_{\max}}]$$

Figure 31. Use of the Matrix Infinity-Norm for Unknown Attack Budgets

Lastly, the situation of unknown defense and attack budgets can also be analyzed with matrix norms. The Frobenius norm is the unbiased choice for this situation since both budgets are unknown and parameterized. Each element of the difference matrix is scrutinized equally with the Frobenius norm. However, nothing prohibits the analyst from using the other matrix norms. If the analyst prefers to bias the analysis in favor of the unknown defense budget, then the matrix 1-norm could be employed. Similarly, if the analyst is biased toward the unknown number of attacks that must be defended, the matrix infinity norm could be used.

To see how matrix norms are computed for the test network with unknown defense and attack budgets, we define three difference matrices, M^{MIN} , M^{MID} and M^{MAX} which denote starting points of minimum, midrange and maximum defense and attack

budgets, respectively. The V-nesting approach is used as the nested defense strategy. The difference matrices for each starting point are presented in Figure 32.

$$M^{MIN} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 12 & 0 & 0 & 0 \\ 0 & 4 & 5 & 16 & 8 & 10 \\ 0 & 4 & 2 & 4 & 5 & 10 \\ 0 & 4 & 5 & 2 & 6 & 0 \\ 0 & 5 & 9 & 2 & 5 & 0 \end{bmatrix} \quad M^{MID} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 3 & 0 & 8 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 3 & 3 & 4 & 13 \\ 0 & 2 & 7 & 7 & 4 & 13 \end{bmatrix}$$

$$M^{MAX} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 1 & 0 & 1 \\ 0 & 2 & 0 & 2 & 3 & 6 \\ 0 & 3 & 4 & 2 & 1 & 0 \end{bmatrix}$$

Figure 32. Difference Matrices for Unknown Defense and Attack Budgets

The associated matrix norms for each difference matrix are presented in Table 11. Note that the matrix norms and the vector norms presented in the previous section produce different values. However, the overall result for the analyst remains the same because the matrix norms associated with M^{MAX} are smaller than the matrix norms for M^{MID} and M^{MIN} . Matrix norm analysis also recommends the choice of a nesting start point of maximum defense and attack budgets when both budgets are unknown.

Table 11. Matrix Norm Analysis at Different Starting Point Instances

Test Network –Defense and Attack Budget Unknown	M^{MIN}	M^{MID}	M^{MAX}
Matrix 1-norm: $\ M\ _1$	33.0	35.0	7.0
Matrix Frobenius-norm: $\ M\ _F$	31.3	24.7	9.5
Matrix ∞ -norm: $\ M\ _\infty$	43.0	32.0	13.0

3. Percentage Gap From Optimality

A different way to measure the distance of a nested defense from the optimal defense is to measure the “cost of prioritization” (Koc & Morton, 2015, p. 591). In this approach, the analyst calculates the percentage gap between the optimal non-nested solution and the nested solution for each DAD instance using the formula in Figure 33.

$$\% \text{ optimality gap} = \frac{(\text{nested objective} - \text{optimal objective})}{\text{optimal objective}} \cdot 100\%$$

Figure 33. Optimality Gap Equation. Adapted from Koc and Morton (2015).

Each nested defense objective function value would be compared to the optimal non-nested objective function value in order to determine the percentage from optimality gap. In the test network, this analysis produces 36 measurements. In order to reduce these 36 measurements to just one measurement, the analyst can use summary statistics. The analyst can calculate the mean and the standard deviation of the percentage of optimality gap. However, the analyst must be cautioned to not make the faulty assumption that an underlying distribution exists on this population of numbers. This set of percentages is not a sample from a larger set and there is no reason to believe that the percentage of optimality gap follows some type of named distribution.

Table 12 summarizes the optimality gap in the situation where both the defense budget and attack budget are unknown. We use the “V” nesting strategy to obtain our results. The optimality gap analysis shows us that the best choice for the nesting start point is the instance of maximum defense and attack budgets. This start point has the smallest mean percentage gap with the tightest standard deviation. In the worst problem instance, this starting point also has the smallest optimality gap.

Table 12. Test Network Optimality Gap Percentages

Defense and Attack Budget Starting point	Optimality Gap Percentages		
	Mean (%)	Std. Dev (%)	Worst (%)
0 (Minimum)	7.10	8.43	28.13
3 (Midrange)	4.32	6.69	22.00
5 (Maximum)	2.10	3.74	13.60

4. Minimax Regret Analysis

Another method to measure the distance from optimality of various nesting strategies would be the decision theory criteria of minimax regret. A common definition of regret is given by Taylor as “the difference between the payoff from the best decision and all other decision payoffs” (2007, p. 519). In our context, each budget scenario instance has a *payoff* of the difference in objective function values between the nested and optimal defenses. Each problem instance can have a different payoff depending on the starting point alternative that is chosen by the analyst. The analyst’s choice of alternatives are the nesting starting point instances of minimum, midrange or maximum defense and/or attack budgets. The goal in minimax regret analysis is to choose the alternative that minimizes the maximum amount of regret for any possible future outcome (Taylor, 2007, p. 519).

For example, we use the test network with unknown defense and attack budgets for our heuristic strategy. The decision matrix in Table 13 gives the difference between the nested defense and optimal non-nested defense objective function values for all future budget scenarios. Each possible future defense and attack budget combination has a payoff that is influenced by the three possible alternatives of beginning the nesting of defenses minimum, midrange or maximum defense and attack budgets. Next, the Table 14 regret matrix is built by examining the payoffs in each column and assigning zero to the smallest value. The regret associated with the other values in each column is computed as the difference between the obtained value and the smallest value. From the regret matrix, we can see that the best choice of nesting starting points is the maximum defense and attack budget alternative because it has the smallest maximum regret.

Table 13. Decision Matrix for Unknown Defense and Attack Budgets

Decision Matrix	Future State (Defense Budget, Attack Budget)																																			
Alternatives (Nesting Start Point Instance)	0,0	0,1	0,2	0,3	0,4	0,5	1,0	1,1	1,2	1,3	1,4	1,5	2,0	2,1	2,2	2,3	2,4	2,5	3,0	3,1	3,2	3,3	3,4	3,5	4,0	4,1	4,2	4,3	4,4	4,5	5,0	5,1	5,2	5,3	5,4	5,5
Minimum budgets	0	0	0	0	0	0	0	0	12	0	0	0	0	4	5	16	8	10	0	4	2	4	5	10	0	4	5	2	6	0	0	5	9	2	5	0
Midrange budgets	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	3	0	8	1	0	0	0	0	1	0	1	3	8	4	13	0	2	7	6	4	13
Maximum budgets	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	2	0	1	1	0	2	0	2	3	6	0	3	4	2	1	0	

Table 14. Regret Matrix for Unknown Defense and Attack Budgets

Regret Matrix	Future State (Defense Budget, Attack Budget)																																			Maximum regret	
Alternatives (Nesting Start Point Instance)	0,0	0,1	0,2	0,3	0,4	0,5	1,0	1,1	1,2	1,3	1,4	1,5	2,0	2,1	2,2	2,3	2,4	2,5	3,0	3,1	3,2	3,3	3,4	3,5	4,0	4,1	4,2	4,3	4,4	4,5	5,0	5,1	5,2	5,3	5,4		5,5
Minimum budgets	0	0	0	0	0	0	0	0	12	0	0	0	0	4	5	16	8	10	0	2	2	4	5	9	0	3	5	0	3	0	0	4	0	0	0		16
Midrange budgets	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	3	0	8	1	0	0	0	0	0	0	0	3	6	1	13	0	0	3	4	3		13
Maximum budgets	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	6	0	1	0	0	0		6

5. Equal Likelihood Analysis

Another method available to the analyst in judging the distance between a nested defense and the optimal defense would be to view each possible defense and attack budget realization as equally likely. In classical decision theory, this type of approach is called the equal likelihood or Laplace criterion (Taylor, 2007, p. 521). This type of decision approach would make sense when the analyst has absolutely no insight about what the realization of the defense or attack budget could be. For each different nesting starting point scenario, the analyst would assign an equal weight to each payoff for all possible defense and attack budget combinations that are unknown. Next, each row alternative is the weighted sum of all payoffs. The smallest sum represents the best outcome, since we are seeking an alternative that is as closest to the optimal solution.

Table 15 shows equal likelihood analysis applied to the test network with unknown defense and attack budgets. Since there are 36 possible combinations of defense and attack budgets for the test network, we assign a weight of 1/36 to each possible payoff in the decision matrix. Each difference value is multiplied by the equally likely

probability of $1/36$ and the sum of all weighted differences is made for each starting point alternative. The smallest sum is preferred choice. Table 15 shows us that the starting point of maximum defense and attack budgets is the best choice.

Table 15. Equal Likelihood Analysis for Unknown Defense and Attack Budgets

Decision Matrix	Future State (Defense Budget, Attack Budget)																																					
Alternatives (Nesting Start Point Instance)	0,0	0,1	0,2	0,3	0,4	0,5	1,0	1,1	1,2	1,3	1,4	1,5	2,0	2,1	2,2	2,3	2,4	2,5	3,0	3,1	3,2	3,3	3,4	3,5	4,0	4,1	4,2	4,3	4,4	4,5	5,0	5,1	5,2	5,3	5,4	5,5	Equal Likelihood	
Minimum budgets	0	0	0	0	0	0	0	0	12	0	0	0	0	4	5	16	8	10	0	4	2	4	5	10	0	4	1	3	8	4	13	0	5	9	2	5	0	3.2777778
Midrange budgets	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	3	0	8	1	0	0	0	4	1	0	1	3	8	4	13	0	2	7	6	4	13	2.0833333	
Maximum budgets	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	2	0	1	0	1	0	2	0	2	3	6	0	3	4	2	1	0	0.7777778		

F. NESTED DEFENSES WITH A SUBOPTIMAL STARTING SCENARIO

In this section, we show that it is possible to find a nested defense that is closer to the optimal solution than any nested defense that begins with an optimal solution to a DAD problem with a parameterized budget. In this approach, the nesting heuristic does not begin with a starting point budget scenario that is equal to the optimal solution. We refer to type of approach to the nested defense DAD problem as *suboptimal nesting*. Suboptimal nesting involves finding nested defenses to parameterized DAD problem instances that are closer to optimal over the entire range of parameterized budgets than any particular nested defense that begins with the optimal solution to the DAD problem instance. The vector norm of differences of the suboptimal nested defense is smaller than the vector norm of any nested defense that begins at an optimal starting point.

Up to this point, we have assumed that a heuristic search for nested defenses must begin with an optimal solution to a nested defense DAD problem instance with a parameterized budget. We now remove that assumption. In this section only, we require that a nested defense objective function value may not equal the optimal defense objective function value for any specific DAD problem instance.

Suboptimal Nested Defense Assumption

$$Z_{ijd}^{\omega} \neq (Z_{ijd}^{\omega})^* \quad \forall \omega \in \Omega \quad (3.3)$$

Proposition 3–4: It is possible that the nested defense closest to the optimal solution may have no instances with objective function values that are equal to the optimal solution.

Proof of proposition 3–4: For an example of an instance where suboptimal nesting is the closest strategy, consider the test network with a defense budget of five units, and a parameterized attack budget. Recall that we use the term *persistent defense* for a known defense budget and an unknown attack budget. An analyst can solve DAD problem instances to find different starting points for persistent defenses. However, the analyst is now interested in finding a suboptimal defense that might be closer to optimal. After trial and error with starting point instances that are suboptimal, the analyst could find a suboptimal nested defense strategy that is overall closer to the optimal non-nested defense in terms of the vector norm than any of the optimal starting point strategies.

The objective function values are shown in Table 16 for each different starting point. The optimal objective function values are shaded green in Table 16 to show that the suboptimal nesting strategy is not optimal for any problem instance. Table 17 shows the persistent defenses for defending the test network against an unknown number of enemy attacks. Figure 34 gives a graphical representation of the data in Table 16. It is not immediately obvious from Table 16 or Figure 34 that the suboptimal nesting strategy is overall the closest nesting strategy to the optimal solution.

Table 16. Objective Function Values for Suboptimal Nesting Example

Nesting Strategy	Attack Budget					
	0	1	2	3	4	5
Optimal Solution (not nested)	20	21	32	41	53	59
Start point: Minimum Attack Budget	20	22	37	62	63	71
Start point: Midrange Attack Budget	23	25	40	41	56	63
Start point: Maximum Attack Budget	25	27	42	43	58	59
Suboptimal Nest	21	23	37	46	55	62

Table 17. Nested Defenses in Test Network for Suboptimal Nesting Example

Nesting Strategy (Defense Budget =5)	Persistent Defenses Chosen
Start point: Minimum Attack Budget	(3, 8) (5, 10) (10, 13)
Start Point: Midrange Attack Budget	(3,8) (9, 13) (10, 11) (10, 13)
Start point: Maximum Attack Budget	(2, 7) (7, 8) (9, 13) (10, 11) (10, 13)
Suboptimal Defense for All Attack Budgets	(8, 12) (3, 8) (9, 10)

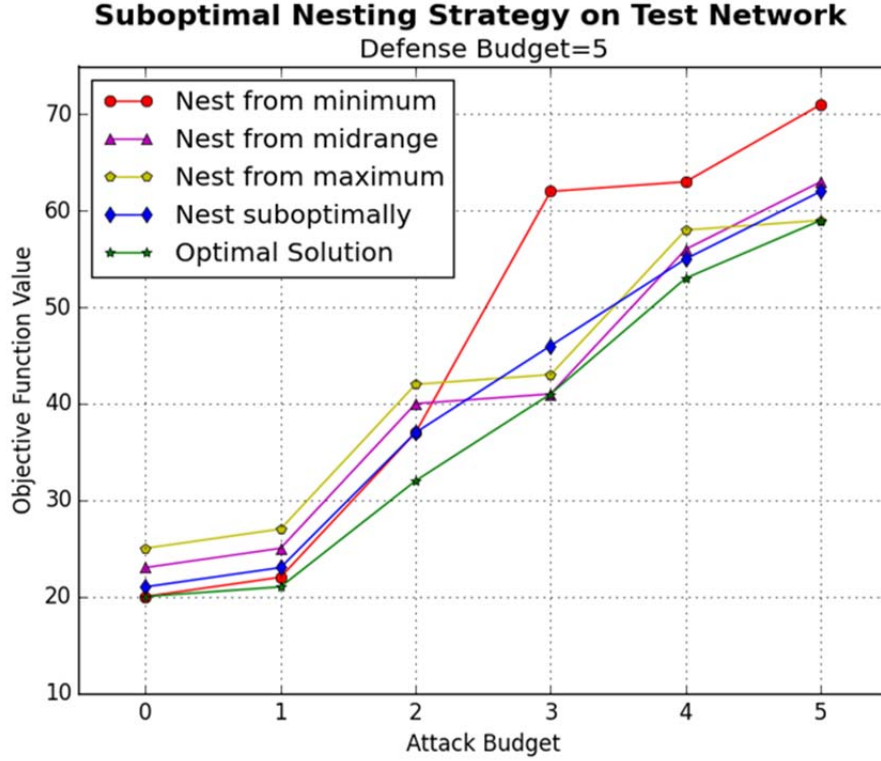


Figure 34. Objective Function Values for Various Nesting Strategies

A comparison of the difference vector norms proves that suboptimal nesting can be closer to optimal than any other nesting strategy. The difference vectors are shown in Figure 35. The definitions of V^{MIN} , V^{MID} , and V^{MAX} are unchanged, and V^{SUB} is defined as the difference between the suboptimal nesting and optimal. The associated norms for these vectors are all given in Table 18. The smallest values for each norm are shaded green in Table 18, and it can easily be seen that the suboptimal nesting strategy has the smallest vector norms. Therefore, it is possible that a suboptimal nesting strategy can be the closest nesting strategy to the optimal solution in a DAD problem instance.

$$M^{MIN} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 12 & 0 & 0 & 0 \\ 0 & 4 & 5 & 16 & 8 & 10 \\ 0 & 4 & 2 & 4 & 5 & 10 \\ 0 & 4 & 5 & 2 & 6 & 0 \\ 0 & 5 & 9 & 2 & 5 & 0 \end{bmatrix} \quad M^{MID} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 3 & 0 & 8 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 3 & 3 & 4 & 13 \\ 0 & 2 & 7 & 7 & 4 & 13 \end{bmatrix}$$

$$M^{MAX} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 1 & 0 & 1 \\ 0 & 2 & 0 & 2 & 3 & 6 \\ 0 & 3 & 4 & 2 & 1 & 0 \end{bmatrix}$$

Figure 35. Difference Vectors for Suboptimal Nesting Example

Table 18. Difference Norms for Suboptimal Nesting Example

Defense Budget= 5 and Attack Budget Unknown	V^{MIN}	V^{MID}	V^{MAX}	V^{SUB}
Vector 1-norm: $\ V\ _1$	49.0	22.0	28.0	18.0
Vector 2-norm: $\ V\ _2$	26.7	10.7	13.8	8.2
Vector ∞ -norm: $\ V\ _\infty$	21.0	8.0	10.0	5.0

Proposition 3–5: A suboptimal nested defense solution with smaller difference norms than all other nested defense solutions with optimal starting points will not always exist for a DAD problem instance with a parameterized budget.

Proof of proposition 3–5. Consider a network where the optimal defenses for an unknown defense budget are also a nested defenses. Nehme and Morton (2010) have proven that such networks exist. A nested defense that is also the optimal defense would have a difference norm of zero. Since no defense can have a better objective function value than the optimal defense in a DAD problem, it would be impossible to find a nested defense that is better than the optimal defense.

We have shown that it is possible to have a nested or persistent defense to a parameterized budget DAD problem instance that does not have any solutions that are

common with the optimal solutions. The problem with searching for a suboptimal nested defense is that finding such a defense is simply an intelligent search by the analyst for a better nested defense than those that are optimal at the nesting starting points. This method relies upon the analyst’s knowledge and experience with the system. Unfortunately, this kind of “trial and error” method for finding a nested defense is nothing more than a heuristic that is limited by the time and talent of the analyst. We need a deliberate way to find the best nested defense for unknown budgets.

G. OPTIMIZING THE BEST NESTED DEFENSE

Our investigation up to this point has been focused on heuristic strategies for finding a nested defense. We now desire an exact procedure to produce the best possible nested defense for a DAD problem. The *best nested defense* is defined as a nested defense that has the smallest difference from the optimal non-nested defenses for a DAD problem instance with parameterized defense and/or attack budgets. The smallest difference can be measured in different ways, which we have previously introduced as the vector difference norms. In this section, we do not place any restrictions on whether nested defenses may or may not equal the optimal defense objective function value. We formulate the nested defense DAD problem as parametric programs with different norms as objective functions.

Our approach to finding the best nested defense to a DAD problem instance fits the framework of a stochastic linear program because some of the problem data is uncertain (Birge & Louveaux, 2011, p. 57). The action of the defender in choosing the best nested defense can be thought of as a “first stage decision” and the worst-case attack chosen by the attacker as well as the subsequent optimal operation of the resulting system can be thought of as “second stage decisions” (Birge & Louveaux, 2011, p. 58). The structure of the nested defense DAD parametric programming problem is reminiscent of the “L-shaped stochastic programs with recourse” that were popularized by Van Slyke and Wets (1969, p. 656). Specifically, the addition of nesting constraints over the set of budget scenarios gives the nested defense DAD problem a “block angular structure” (Birge & Louveaux, 2011, p. 183), except our formulation is somewhat more complicated

than the models seen in the literature. The extra complexity of the nested defense DAD problem resides in the fact that the outer minimization and maximization operators are both over integer variables. Our approach also departs classic stochastic programming theory because we do not represent the uncertain budgetary data as a random variable.

Sets:

$\omega \in \Omega$ Uncertain defense and/or attack budget scenarios

Data

$attackbudget^\omega$ Attack budget in scenario ω .
 $defensebudget^\omega$ Defense budget in scenario ω .
 $optimal^\omega$ Optimal (non-nested) objective function value for budget ω .

Decision Variables

S_n^ω Amount of unfulfilled demand at node n in budget scenario ω .
 W_{ijd}^ω Binary; 1 if edge (i, j) chooses defense option d in scenario ω .
 X_{ij}^ω Binary; 1 if edge (i, j) is attacked, 0 otherwise in scenario ω .
 Y_{ijd}^ω Amount of flow on arc (i, j) for defense d in budget scenario ω .

1. Best Nested Defense as Measured by 1-Norm

An intuitive way to measure the best nested defense is to add up all of the differences between a nested defense and the optimal defense for all possible budget scenarios. The best nested defense would then have the smallest sum over the budget scenarios. The 1-norm formulation of the nested defense DAD problem will find the best nested defense for a network with parameterized budget scenarios in terms of the sum of all deviations from optimality.

DAD Problem with Nested Defense Formulation (1-norm)

$$\|V\|_1 = \sum_{\omega \in \Omega} \sum_{d \in D} \sum_{(i,j) \in E} (c_{ijd} + q_{ijd} \cdot X_{ij}^\omega) \cdot Y_{ijd}^\omega + (c_{jid} + q_{jid} \cdot X_{ij}^\omega) \cdot Y_{jid}^\omega + \sum_{n \in N} S_n^\omega \cdot p_n - optimal^\omega, (3.4a)$$

Objective Function:

$$\min_{W_{ijd}^\omega} \max_{X_{ij}^\omega} \min_{Y_{ijd}^\omega, S_n^\omega} \|V\|_1, \quad (3.4b)$$

Subject to:

$$\sum_{d \in D} \sum_{(n,j) \in A} Y_{njd}^\omega - \sum_{d \in D} \sum_{(i,n) \in A} Y_{ind}^\omega - S_n^\omega \leq demand_n \quad \forall n \in N, \omega \in \Omega, \quad (3.4c)$$

$$\sum_{(i,j) \in E} (Y_{ijd}^\omega + Y_{jid}^\omega) \leq u_{ijd} \cdot W_{ijd}^\omega \quad \forall (i,j) \in E, d \in D, \omega \in \Omega, \quad (3.4d)$$

$$\sum_{(i,j) \in A} r_{ij} \cdot X_{ij}^\omega \leq attack_budget^\omega \quad \forall \omega \in \Omega, \quad (3.4e)$$

$$\sum_{d \in D: (i,j) \in E} h_{ijd} \cdot W_{ijd}^\omega \leq defense_budget^\omega \quad \forall \omega \in \Omega, \quad (3.4f)$$

$$\sum_{d \in D} W_{ijd}^\omega = 1 \quad \forall (i,j) \in E, \omega \in \Omega, \quad (3.4g)$$

$$W_{ijd}^\omega \leq W_{ijd}^{\omega+1} \quad \forall (i,j) \in E, d \in D: d \neq d_0, \omega \in \Omega, \quad (3.4h)$$

$$Y_{ijk}^\omega \geq 0 \quad \forall (i,j) \in A, d \in D, k \in K, \omega \in \Omega, \quad (3.4i)$$

$$S_n^\omega \geq 0 \quad \forall n \in N, \omega \in \Omega, \quad (3.4j)$$

$$X_{ij}^\omega \in \{0,1\} \quad \forall (i,j) \in E, \omega \in \Omega, \quad (3.4k)$$

$$W_{ijd}^\omega \in \{0,1\} \quad \forall (i,j) \in E, d \in D, \omega \in \Omega. \quad (3.4l)$$

Equation (3.4a) defines the 1-norm as the sum across all budget scenarios of the difference between the nested defense and the optimal defense. Expression (3.4b) is the tri-level optimization objective to minimize the 1-norm for the DAD problem instance. Equation (3.4c) represents the balance of flow constraints for each node in the network under each budget scenario, with an elastic variable (S) compensating for unfulfilled demand. Equations (3.4d) are the bi-directional capacity constraints for each edge in the network, forcing total flow on an edge to be less than the maximum capacity for the edge under a defense option chosen for each budget scenario. Equation (3.4e) is a budget constraint on the attacker to ensure that the attacker can only attack as many edges as he or she can afford for each budget scenario. Equation (3.4f) is a budget constraint on the defender which ensures the amount of network defense stay within the budget scenario. Equation (3.4g) ensures each edge in the network can only have one defense option chosen for each budget scenario. Equation (3.4h) ensures the defenses chosen in one

budget scenario are also chosen in the next budget scenario. Equations (3.4i)-(3.4l) enforce variable types.

The system of Equations (3.4) is a linear program, even though it might not appear to be initially. The attack (X) and flow (Y) decision variables are multiplied together, but one variable is maximized, and the other variable is minimized. Decomposition of the problem formulation results in a linear program. Nested decomposition procedures are similar to those discussed in Chapter I.

Next, we perform decomposition of the nested defense DAD problem with 1-norm objective. The first part of the decomposition is the AD subproblem. The AD subproblem primal version has a max-min objective function. In order to simplify the problem, we take the dual of the inner minimization operator model, since the flow decision variables (Y) are continuous. We define ‘modular decomposition’ of the AD subproblem to be the solution to a series of AD subproblems that each pertain to a specific defense and attack budget scenario (ω).

Additional Decision Variables:

α_{ijd}^ω	Dual variable for edge (i, j) with defense option d for budget scenario ω
π_n^ω	Dual variable for node n for budget scenario ω .
Z_{DUAL}^ω	AD subproblem objective for each scenario ω .
Z_{SUB_TOTAL}	Sum of all AD subproblem objectives in the set of budget scenarios Ω .

Nested Defense AD Dual Subproblem, 1-Norm Objective:

$$Z_{SUB_TOTAL} = \max_{X_{ij}^\omega, \alpha_{ijd}^\omega, \pi_n^\omega} \sum_{\omega \in \Omega} Z_{DUAL}^\omega, \quad (3.5a)$$

Subject to AD Subproblem Modular Decompositions:

$$Z_{DUAL}^\omega = - \sum_{n \in N} \pi_n^\omega \cdot demand_n - \sum_{(i,j) \in E} \sum_{d \in D} u_{ijd} \cdot \alpha_{ijd}^\omega \cdot \hat{W}_{ijd}^\omega - optimal^\omega \quad \forall \omega \in \Omega, \quad (3.5b)$$

Subject to:

$$-\pi_i^\omega + \pi_j^\omega - \alpha_{ijd}^\omega \Big|_{i < j} - \alpha_{jid}^\omega \Big|_{i > j} \leq c_{ijd} + q_{ijd} \cdot \left(X_{ij}^\omega \Big|_{i < j} + X_{ji}^\omega \Big|_{i > j} \right) \quad \forall (i, j) \in E, d \in D, \omega \in \Omega, \quad (3.5c)$$

$$\pi_n^\omega \leq p_n \quad \forall n \in N, \omega \in \Omega, \quad (3.5d)$$

$$\sum_{(i,j) \in A} r_{ij} \cdot X_{ij}^\omega \leq \text{attack_budget}^\omega \quad \forall \omega \in \Omega, \quad (3.5e)$$

$$-\pi_n^\omega \geq 0 \quad \forall n \in N, \omega \in \Omega, \quad (3.5f)$$

$$-\alpha_{ijd}^\omega \geq 0 \quad \forall (i, j) \in A, d \in D, \omega \in \Omega, \quad (3.5g)$$

$$X_{ij}^\omega \in \{0, 1\} \quad \forall (i, j) \in E, \omega \in \Omega, \quad (3.5h)$$

$$\sum_{(i,j) \in E} \sum_{\substack{\omega \in \Omega: \\ X_{ijk}^\omega = 0}} X_{ij}^\omega + \sum_{(i,j) \in E} \sum_{\substack{\omega \in \Omega: \\ X_{ijk}^\omega = 1}} (1 - X_{ij}^\omega) \geq 1. \quad (3.5i)$$

Expression (3.5a) of the AD decomposition uses a dual Integer Linear Program (ILP) formulation to transform the max-min structure of the AD subproblem to a maximization problem. Expression (3.5a) is the beginning of the modular decomposition of the original problem, because the objective is a summation of a series of AD subproblems for all of the budget scenarios. Equation (3.5b) shows that each budget scenario objective is a difference between the nested defense solution and the optimal solution. The optimal solution for each budget scenario was computed with the formulation in Chapter I and is included as data in this model. Equation (3.5c) is the constraint for each undirected edge under each budget scenario. Equation (3.5d) is the dual constraint for each node of the network for each budget scenario. Equation (3.5e) is the restatement of Equation (3.4e) in the decomposition. Equations (3.5f)-(3.5h) define variable types. Equation (3.5i) are the SEC needed for decomposition of the binary variables when attacks are repeated in iterations of the master problem. The subscript “ k ” in Equation (3.5i) refers to the iteration of the decomposition for the master problem.

The system of Equations (3.5a-i) are decoupled over all of the budget scenarios (ω). The decoupling can be observed by noting that the defense variables (W) have fixed values, which is symbolized by the hats on these variables. There is no constraint within the AD subproblem that connects any budget scenario (ω) to any other budget scenario. Thus, each budget scenario (ω) is independent in the AD subproblem, which allows for the commutation of the summation and maximization operators in Equation (3.5a).

The nested defense DAD decomposition master problem takes the attacks (X) for each budget scenario (ω) as well as the AD subproblem objective function as an upper bound. The master problem links together all of the budget scenarios over a series of iterations in the outer decomposition. The result of each iteration of the nested defense DAD master problem is a potential lower bound and a set of defenses chosen (W) to begin the next iteration of AD subproblems.

Additional Sets:

$k \in K$ Decomposition iterations

Additional Decision Variables (units):

S_{nk}^ω Amount of unfulfilled demand at node n in scenario ω for iteration k .
 X_{ijk}^ω Binary; 1 if edge (i, j) is attacked in budget scenario ω for iteration k .
 Y_{ijk}^ω Amount of flow on arc (i, j) for defense d in scenario ω for iteration k .
 Z_{DAD}^ω Difference of nested & optimal objective function values for scenario ω .
 Z_{DAD_TOTAL} DAD master problem objective function value

Nested Defense DAD Master Problem, 1-Norm Objective:

$$Z_{DAD_TOTAL} = \min_{W_{ijd}^\omega, Y_{ijk}^\omega, S_{nk}^\omega} \sum_{\omega \in \Omega} Z_{DAD}^\omega, \quad (3.5j)$$

Subject to DAD modular decompositions:

$$Z_{DAD}^\omega \geq \sum_{d \in D} \sum_{(i,j) \in E} \left(c_{ijd} + (q_{ijd} \cdot \hat{X}_{ijk}^\omega) \right) \cdot Y_{ijk}^\omega + \left(c_{jid} + (q_{jid} \cdot \hat{X}_{ijk}^\omega) \right) \cdot Y_{jkd}^\omega - optimal^\omega \forall \omega \in \Omega, \quad (3.5k)$$

$$\sum_{d \in D: (i,j) \in E} h_{ijd} \cdot W_{ijd}^\omega \leq defense_budget^\omega \quad \forall \omega \in \Omega, \quad (3.5l)$$

$$\sum_{d \in D} \sum_{(n,j) \in A} Y_{njd}^\omega - \sum_{d \in D} \sum_{(i,n) \in A} Y_{ink}^\omega - S_{nk}^\omega \leq demand_n \quad \forall n \in N, k \in K, \omega \in \Omega, \quad (3.5m)$$

$$\sum_{(i,j) \in E} (Y_{ijk}^\omega + Y_{jkd}^\omega) \leq u_{ijd} \cdot W_{ijd}^\omega \quad \forall (i,j) \in E, d \in D, k \in K, \omega \in \Omega, \quad (3.5n)$$

$$\sum_{d \in D} W_{ijd}^\omega = 1 \quad \forall (i,j) \in E, \omega \in \Omega, \quad (3.5o)$$

$$W_{ijd}^\omega \leq W_{ijd}^{\omega+1} \quad \forall (i,j) \in E, d \in D : d \neq d_0, \omega \in \Omega, \quad (3.5p)$$

$$W_{ijd}^\omega \in \{0, 1\} \quad \forall (i,j) \in E, d \in D, \omega \in \Omega, \quad (3.5q)$$

$$S_{nk}^{\omega} \geq 0 \quad \forall n \in N, k \in K, \omega \in \Omega, \quad (3.5r)$$

$$Y_{ijk}^{\omega} \geq 0 \quad \forall (i, j) \in A, d \in D, k \in K, \omega \in \Omega. \quad (3.5s)$$

Expression (3.5j) is the decomposition master problem, which minimizes the sum of differences between the nested defense and optimal defense, for all budget scenarios. Equation (3.5k) represents the difference between the nested defense objective function value and the optimal value for each of the budget scenarios. Equation (3.5l) is the budget constraint on the defender for each of the budget scenarios. Equation (3.5m) represents balance of flow on the network for each of the budget scenarios. Equation (3.5n) enforces the maximum flow on each edge of the network when the defense option is chosen for each of the budget scenarios. Equation (3.5o) allows one defense option for each edge, for each budget scenario. Equations (3.5p) are the nesting constraints, which force the defenses chosen in a budget scenario (ω) to also be chosen in the next budget scenario ($\omega+1$). Equations (3.5q-s) enforce variable types.

2. Best Nested Defense as Measured by Infinity-Norm

There are also other ways that an analyst may choose to define the best nested defense. The analyst may desire that the best nested defense must have the smallest possible difference from the optimal solution for any budget scenario. With this view, the objective function value for the nested defense may differ from the optimal value for many budget scenarios, but the largest difference is as small as possible. The infinity norm formulation of the nested defense DAD problem will find the best nested defense for a network with parameterized budget scenarios by keeping the largest difference between nested defense and optimal defense as small as possible. If the analyst desires to find the best infinity-norm nested defenses, he or she can modify the nested defense DAD problem as follows:

Nested Defense DAD Problem Formulation (Infinity-Norm)

$$\|V\|_{\infty} = \max_{\omega \in \Omega} \sum_{d \in D} \sum_{(i,j) \in E} (c_{ijd} + q_{ijd} X_{ij}^{\omega}) \cdot Y_{ijd}^{\omega} + (c_{jid} + q_{jid} X_{ij}^{\omega}) \cdot Y_{jid}^{\omega} + \sum_{n \in N} S_n^{\omega} \cdot p_{n-optimal}^{\omega}, \quad (3.6a)$$

Objective function (Infinity-norm):

$$\min_{W_{ij}^\omega} \max_{X_{ij}^\omega} \min_{Y_{ij}^\omega, S_n^\omega} \|V\|_\infty. \quad (3.6b)$$

Equation (3.6a) defines the infinity norm for the nested defense DAD problem and would replace Equation (3.4a) in the formulation of the nested defense DAD problem. Equation (3.6a) finds the maximum difference between the nested defense objective function value and the optimal defense objective function value over the set of all possible budget scenarios. Expression (3.6b) replaces (3.4b) in our formulation. Expression (3.6b) is a four level optimization problem that seeks to minimize the maximum difference between the nested defense and the optimal defense. The AD sub-problem decomposition constraints of Equations (3.4c–l) remain for the infinity norm formulation. The decomposition of the infinity norm nested defense DAD problem has two changes from the 1-norm decomposition problem presented in the previous section.

DAD Master Problem, Infinity-Norm Objective:

$$\min_{W, Y} Z_{DAD_TOTAL}, \quad (3.6c)$$

Add the following constraint for DAD modular decomposition:

$$Z_{DAD_TOTAL} \geq Z_{DAD}^\omega \quad \forall \omega \in \Omega. \quad (3.6d)$$

Equation (3.6c) replaces Equation (3.5j) as the objective for the decomposition master problem. The decomposition problem as seen in Equations (3.5k–r) are the same in the infinity norm case, and Equation (3.6d) is added as an additional constraint to ensure the master problem objective function value is at least as large as the biggest difference between nested defense and optimal defense. Equation (3.6c) works along with Equation (3.6d) to minimize the maximum difference between nested defense and optimal defense over the set of all budget scenarios.

3. Best Nested Defense as Measured by 2-Norm

The 2-norm is desirable as an objective for the parametric programming problem because it can be thought of increasing the amount of penalty to the objective function as the difference between the nested defense and optimal defense grows. However, the 2-norm formulation of the nested defense DAD problem is more challenging to implement because it is a nonlinear quadratic program. Therefore, our formulation will use the square of the 2-norm in order to ensure we have a convex optimization problem. In our results, we report the square root of the solution to the quadratic parametric program in order to obtain the actual 2-norm value.

Nested Defense DAD Problem Formulation (Squared 2-Norm)

$$\|V\|_2^2 = \sum_{\omega \in \Omega} \left(\sum_{d \in D} \sum_{(i,j) \in E} (c_{ijd} + q_{ijd} X_{ij}^\omega) \cdot Y_{ijd}^\omega + (c_{jid} + q_{jid} X_{ij}^\omega) \cdot Y_{jid}^\omega + \sum_{n \in N} S_n^\omega \cdot p_n - \text{optimal}^\omega \right)^2, \quad (3.7a)$$

Objective function (Squared 2-Norm):

$$\min_{W_{ij}^\omega} \max_{X_{ij}^\omega} \min_{Y_{ijd}^\omega, S_n^\omega} \|V\|_2^2. \quad (3.7b)$$

Equation (3.7a) defines the squared 2 norm for the nested defense DAD problem. Expression (3.7b) is the tri-level optimization objective to minimize the squared 2-norm for the nested defense DAD problem instance. The AD sub-problem decomposition constraints of Equations (3.4c–l) remain the same for the squared 2-norm formulation. Equation (3.7a) replaces (3.4a) and (3.7b) replaces (3.4b) when the squared 2-norm is the objective. The decomposition of the squared 2-norm nested defense DAD problem has two changes from the 1-norm decomposition problem presented in the previous section.

AD Dual Subproblem, Squared 2-Norm Objective:

$$Z_{SUB_TOTAL} = \sum_{\omega \in \Omega} \left(\max_{X_{ij}^\omega, \alpha_{ijd}^\omega, \pi_n^\omega} Z_{DUAL}^\omega \right)^2. \quad (3.7c)$$

Equation (3.7c) replaces Equation (3.5a) from the 1-norm AD subproblem formulation. Equation (3.7c) is the sum of the squared values of a series of AD subproblems, one for each budget scenario. The constraints in the AD dual subproblem for the 2-norm objective function are the same as those for the 1-norm AD dual ILP subproblem that are found in Equations (3.5b–i).

DAD Master Problem, Squared 2-Norm Objective:

$$Z_{DAD_TOTAL} = \sum_{\omega \in \Omega} \min_{W_{jkd}^{\omega}, Y_{jkd}^{\omega}, S_{nk}^{\omega}} \left(Z_{DAD}^{\omega} \right)^2. \quad (3.7d)$$

Equation (3.7d) replaces Equation (3.5j) from the 1-norm formulation. Equation (3.7d) makes the master problem a quadratic program. The remaining constraints are unchanged from the 1-norm DAD master problem, found in Equations (3.5l–r).

4. Implementation on Test Network

Our formulation can find the best possible nested defense in terms of the 1-norm, infinity norm, or squared 2 norm on the test network. The following examples illustrate how our formulations can find better nested defenses to DAD problem instances than the heuristic approaches.

a. 1-Norm Formulation Example

The first example is the 1-norm formulation for the nested defense DAD problem on the test network when the defense budget is parameterized, but the attack budget is known to be two units. Tables 19 and 20 summarize the different nested defenses chosen and objective function values for each nesting strategy. Table 17 shows us that the first defense chosen by the 1-norm parametric program is the same as the minimum starting point heuristic, but the other defenses chosen are different. Our formulation is able to exploit a weakness of the heuristic. The heuristic does not explore all equivalent optimal solutions every time the parameterized budget is incremented by one unit.

Table 19. Objective Function Values in 1-Norm Nesting Example

Nesting Strategy (Attack Budget = 2)	Defense Budget					
	0	1	2	3	4	5
Optimal Solution (not nested)	62	50	45	45	37	32
Start point: Minimum Defense Budget	62	50	47	47	39	38
Start point: Midrange Attack Budget	62	62	47	45	39	38
Start point: Maximum Attack Budget	62	62	47	45	38	32
1-norm Nesting Strategy	62	50	47	47	38	32

Table 20. Nested Defenses Chosen in 1-Norm Nesting Example

Nesting Strategy Attack Budget =2	Defenses Chosen at Defense Budget Level					
	0	1	2	3	4	5
Start point: Minimum Defense Budget	None	(2,7)	(2,7) (7,8)	(2,7) (7,8)	(2,7) (7,8) (5,10)	(2,7) (7,8) (5,10) (8,12)
Start Point: Midrange Defense Budget	None	None	(5,10)	(5,10) (7,8)	(5,10) (7,8) (2,7)	(5,10) (7,8) (2,7) (8,12)
Start point: Maximum Defense Budget	None	None	(5,10)	(5,10) (8,12)	(5,10)(8,12) (2,7)	(5,10) (8, 12) (2,7) (11, 15)
1-norm Nesting Strategy	None	(2,7)	(2,7) (8,12)	(2,7) (8,12)	(2,7) (8,12) (5,10)	(2,7) (8,12) (5,10) (13,14)

Figure 36 presents a graphical display of the information contained in Table 19. In Figure 36, the blue diamonds that represent the 1-norm nested defenses appear to be the closest to the optimal solutions shown by the green stars. In order to prove that the 1-norm is the nested defense that is closest to the optimal non-nested defense in this example instance, we show the difference vectors in Figure 37. We use the notation V^{MIN} , V^{MID} , and V^{MAX} for the difference vectors associated with the heuristic starting points of minimum, midrange and maximum defense budgets, respectively. We define V^{L1} for the difference vector associated with the 1-norm parametric program. In Table 21, we compute the various difference norms associated with the difference vectors. Even though our parametric program in this example had the minimization of the 1-norm as its

objective, we observe in Table 21 that the resulting nested defense has the smallest 1-norm, 2-norm and infinity norm of all of the nesting strategies tried in the example.

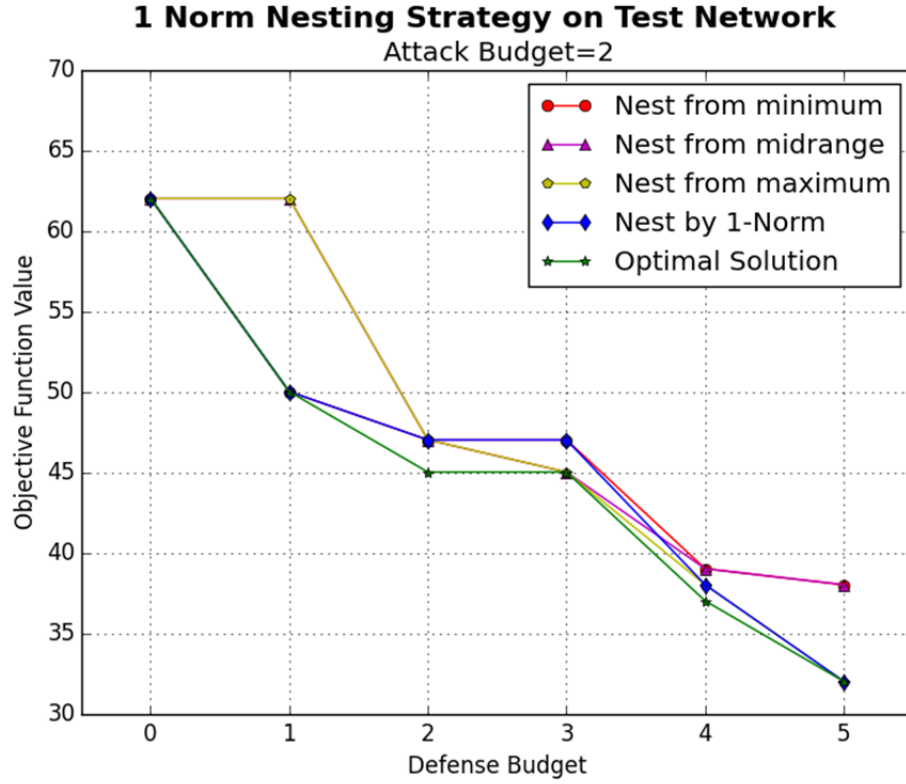


Figure 36. Test Network Nested Defense with 1-Norm Nesting Example

$$V = \begin{bmatrix} \delta_{0,2} \\ \delta_{1,2} \\ \delta_{2,2} \\ \delta_{3,2} \\ \delta_{4,2} \\ \delta_{5,2} \end{bmatrix} \quad V^{MIN} = \begin{bmatrix} 0 \\ 0 \\ 2 \\ 2 \\ 2 \\ 6 \end{bmatrix} \quad V^{MID} = \begin{bmatrix} 0 \\ 12 \\ 2 \\ 0 \\ 2 \\ 6 \end{bmatrix} \quad V^{MAX} = \begin{bmatrix} 0 \\ 12 \\ 2 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad V^{L1} = \begin{bmatrix} 0 \\ 0 \\ 2 \\ 2 \\ 1 \\ 0 \end{bmatrix}$$

Figure 37. Difference Vectors for 1-Norm Nesting Example

Table 21. Difference Norms for 1-Norm Nesting Example

Defense Budget Unknown Attack Budget =2	V^{MIN}	V^{MID}	V^{MAX}	V^{L1}
Vector 1-norm: $\ V\ _1$	12	22	15	5
Vector 2-norm: $\ V\ _2$	6.93	13.71	12.21	3
Vector ∞ -norm: $\ V\ _\infty$	6	12	12	2

b. Infinity Norm Formulation Example

An example of nesting defenses based on the infinity norm formulation considers the test network with an unknown defense budget and attack budget fixed at four units. Tables 22 and 23 summarize the different nested defenses chosen and objective function values for each nesting strategy. Table 23 shows that the first defense chosen by the infinity norm parametric program is the same as the minimum and maximum heuristics.

Table 22. Objective Function Values for Infinity-Norm Nesting Example

Nesting Strategy (Attack Budget = 4)	Defense Budget					
	0	1	2	3	4	5
Optimal Solution (not nested)	113	82	74	66	58	53
Start point: Minimum Defense Budget	113	82	74	74	66	53
Start point: Midrange Attack Budget	113	113	81	66	58	57
Start point: Maximum Attack Budget	113	82	82	69	61	53
Infinity-Norm Stochastic Program	113	82	79	71	63	55

Table 23. Nested Defenses Chosen for Infinity-Norm Nesting Example

Nesting Strategy Att. Budget =4	Defenses Chosen at Defense Budget Level					
	0	1	2	3	4	5
Start point: Minimum Defense Budget	None	(10,13)	(10,13) (9,13)	(10,13) (9,13)	(10,13) (9,13) (3,8)	(10,13) (9,13) (3,8) (8,12)
Start Point: Midrange Defense Budget	None	None	(5,10)	(5,10) (10,11)	(5,10) (10,11) (1,5)	(5,10) (10,11) (1,5) (6,10)
Start point: Maximum Defense Budget	None	(10,13)	(10,13)	(10,13) (3,8)	(10,13) (3,8) (8,12)	(10,13) (3,8) (8,12) (9,13)
Infinity-norm Stochastic Program	None	(10,13)	(10,13) (7,8)	(10,13) (7,8) (2,7)	(10,13) (7,8) (2,7) (8,12)	(10,13) (7,8) (2,7) (8,12) (9,13)

Figure 38 presents a graphical display of the information contained in Table 22. The blue diamonds that represent the infinity norm nested defenses do not necessarily appear to be the closest to the optimal solutions shown by the green stars. In order to prove that the infinity-norm is the nested defense that is closest to the optimal non-nested defense in this example instance, we show the difference vectors in Figure 39. We define V^{L^∞} as the difference vector associated with the infinity norm formulation. In Table 24, we compute the various difference norms for each nesting strategy. Table 24 shows that the infinity norm objective has the smallest infinity norm of all the nesting strategies in this example problem instance. Surprisingly, even though our formulation in this example had the minimization of the infinity norm as its objective, we observe that the resulting defense also has the smallest 2-norm all of the nesting strategies.

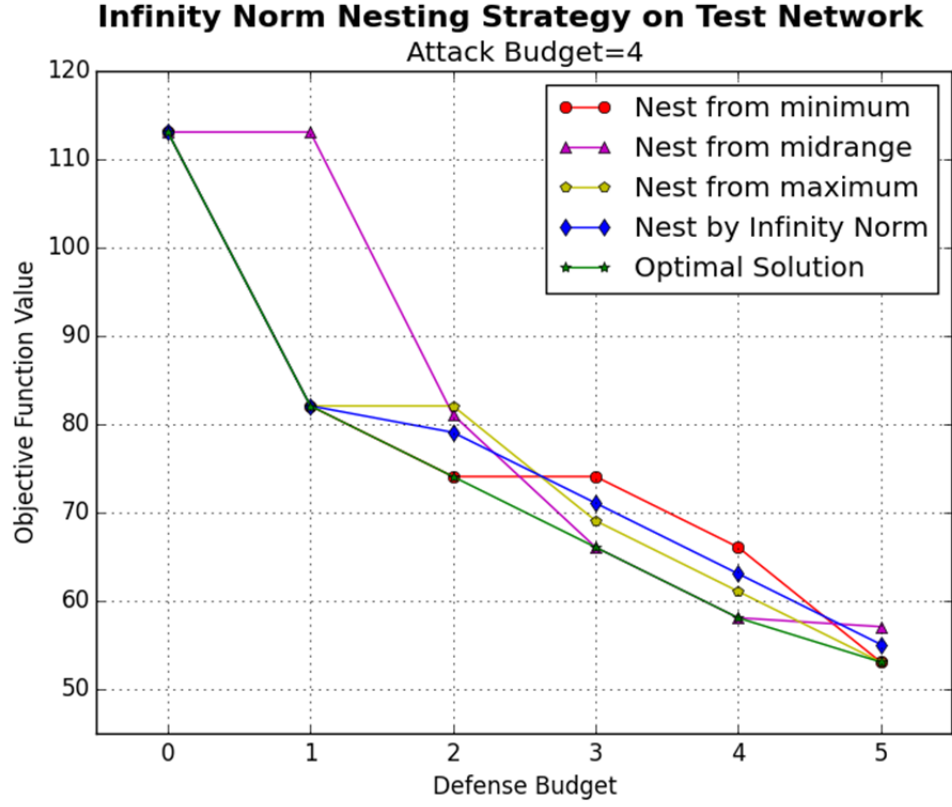


Figure 38. Test Network Nested Defense with Infinity-Norm Nesting Example

$$V = \begin{bmatrix} \delta_{0,4} \\ \delta_{1,4} \\ \delta_{2,4} \\ \delta_{3,4} \\ \delta_{4,4} \\ \delta_{5,4} \end{bmatrix} \quad V^{MIN} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 8 \\ 8 \\ 0 \end{bmatrix} \quad V^{MID} = \begin{bmatrix} 0 \\ 31 \\ 7 \\ 0 \\ 0 \\ 4 \end{bmatrix} \quad V^{MAX} = \begin{bmatrix} 0 \\ 0 \\ 8 \\ 3 \\ 3 \\ 0 \end{bmatrix} \quad V^{L\infty} = \begin{bmatrix} 0 \\ 0 \\ 5 \\ 5 \\ 5 \\ 2 \end{bmatrix}$$

Figure 39. Difference Vectors for Infinity-Norm Nesting Example

Table 24. Difference Norms for Infinity-Norm Nesting Example

Defense Budget Unknown Attack Budget = 4	V^{MIN}	V^{MID}	V^{MAX}	$V^{L\infty}$
Vector 1-norm: $\ V\ _1$	16	42	14	17
Vector 2-norm: $\ V\ _2$	11.31	32.03	9.06	8.89
Vector ∞ -norm: $\ V\ _\infty$	8	31	8	5

c. Squared 2-Norm Formulation Example

An example based on the squared 2-norm formulation considers the test network with an unknown defense budget and attack budget fixed at five units. Tables 25 and 26 summarize the different nested defenses chosen and objective function values for each nesting strategy. We observe the defenses chosen and the objective function value of the squared 2-norm formulation is the same as the maximum defense starting point heuristic. In this case, the heuristic was able to find the nested defense that is closest to the optimal non-nested defense as measured by the squared 2-norm. Our formulation is also able to prove that these defenses are the closest to the optimal non-nested defenses.

Table 25. Objective Function Values for Squared 2-Norm Nesting Example

Nesting Strategy (Attack Budget = 5)	Defense Budget					
	0	1	2	3	4	5
Optimal Solution (not nested)	131	104	87	79	65	59
Start point: Minimum Defense Budget	131	104	96	88	79	64
Start point: Midrange Attack Budget	131	131	87	79	71	67
Start point: Maximum Attack Budget	131	104	97	80	65	59
Squared 2-Norm Stochastic Program	131	104	97	80	65	59

Table 26. Nested Defenses Chosen for Squared 2-Norm Nesting Example

Att. Budget =5 Nesting Strategy	Defenses Chosen at Defense Budget Level					
	0	1	2	3	4	5
Start point: Minimum Defense Budget	None	(10,13)	(10,13) (9,13)	(10,13) (9,13) (8,12)	(10,13) (9,13) (8,12) (7,8)	(10,13) (9,13) (8,12) (7,8) (2,7)
Start Point: Midrange Defense Budget	None	None	(5,10)	(5,10) (8,12)	(5,10) (8,12) (1,5)	(5,10) (8,12) (1,5) (11,15)
Start point: Maximum Defense Budget	None	(10,13)	(10,13) (10,11)	(10,13) (10,11) (7,8)	(10,13) (10,11) (7,8) (2,7)	(10,13) (10,11) (7,8) (2,7) (9,13)
Squared 2-norm Nesting Strategy	None	(10,13)	(10,13) (10,11)	(10,13) (10,11) (7,8)	(10,13) (10,11) (7,8) (2,7)	(10,13) (10,11) (7,8) (2,7) (9,13)

Figure 40 presents a graphical display of the information contained in Table 23. The blue diamonds that represent the 2-norm nested defenses appear to be the closest to

the optimal solutions shown by the green stars. In order to prove that the 2-norm is the best nested defense in this example instance, we show the difference vectors in Figure 41. We introduce V^{L2} for the difference vector associated with the squared 2-norm stochastic program. In Table 27, we compute the various difference norms associated with each nesting strategy. In Table 27 we report the actual 2-norm, which is the square root of the objective function value of the stochastic program. Table 27 shows that squared 2 norm objective and the heuristic of starting at maximum defenses both have the smallest 2-norm of all the nesting strategies in this example. Surprisingly, even though our parametric program in this example has minimization of the squared 2-norm as the objective, it also has the smallest 1-norm and infinity norm.

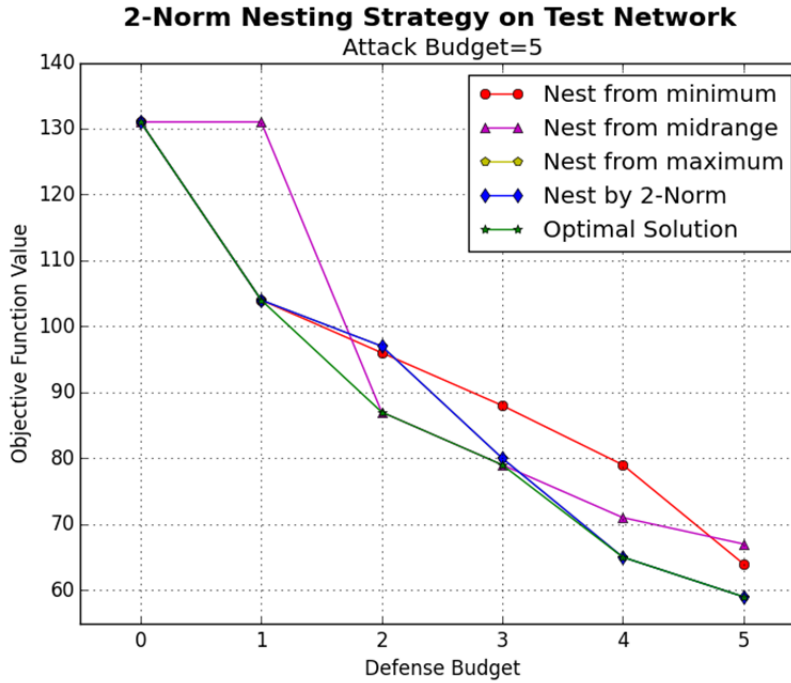


Figure 40. Test Network Nested Defense with Squared 2 Norm Objective

$$V = \begin{bmatrix} \delta_{0,5} \\ \delta_{1,5} \\ \delta_{2,5} \\ \delta_{3,5} \\ \delta_{4,5} \\ \delta_{5,5} \end{bmatrix} \quad V^{MIN} = \begin{bmatrix} 0 \\ 0 \\ 9 \\ 9 \\ 14 \\ 5 \end{bmatrix} \quad V^{MID} = \begin{bmatrix} 0 \\ 27 \\ 0 \\ 0 \\ 6 \\ 8 \end{bmatrix} \quad V^{MAX} = \begin{bmatrix} 0 \\ 0 \\ 10 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad V^{L2} = \begin{bmatrix} 0 \\ 0 \\ 10 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Figure 41. Difference Vectors for 2-Norm Nesting Example

Table 27. Difference Norms for 2-Norm Nesting Example

Defense Budget Unknown Attack Budget = 4	V^{MIN}	V^{MID}	V^{MAX}	V^{L2}
Vector 1-norm: $\ V\ _1$	37	41	11	11
Vector 2-norm: $\ V\ _2$	19.57	28.79	10.04	10.04
Vector ∞ -norm: $\ V\ _\infty$	14	27	10	10

H. NESTED DEFENSES IN MAXIMUM FLOW PROBLEMS

In a maximum flow network problem, the nesting of defenses against worst-case attack presents a different kind of problem. In this case, the network has an infinite supply of items at the source of the network, and an infinite demand for items at the destination of the network. The edges in the network are capacitated in the number of units that can travel over them. The DAD maximum flow problem is adapted from the basic maximum flow linear program that can be found in texts such as Ahuja et al. (1993, p. 168) and the network interdiction problem of Wood (1993).

DAD Maximum Flow Problem Formulation:

Sets:

$n \in N$	Nodes of the network (Note: i and j are used as aliases for n) (Note: s and t are the source and terminal nodes in the network)
$(i, j) \in E$	Undirected edges of the network
$(i, j) \in A$	Directed arcs of the network
$d \in D$	Defense plans available for undirected edges (i, j)

Data:

u_{ijd}	Maximum capacity of undirected edge (i, j) under defense plan d .
q_{ijd}	Penalty cost for travel on attacked arc (i, j) with defense plan d .
r_{ij}	Cost to attack undirected edge (i, j) .
h_{ijd}	Cost to protect or build undirected edge (i, j) under defense plan d .
$attack_budget$	Maximum attack budget.
$defense_budget$	Maximum defense budget.

Decision Variables (units):

V	Amount of maximum flow from source to destination (commodity units)
W_{ijd}	Binary; 1 if undirected edge (i, j) chooses defense option d , 0 otherwise.
X_{ij}	Binary; 1 if edge (i, j) is attacked, 0 otherwise.
Y_{ijd}	Amount of flow on directed arc (i, j) for defense d . (commodity units)

Formulation:

$$\max_{W_{ijd}} \min_{X_{ij}} \max_{V, Y_{ijd}} \left(V - \sum_{(i,j) \in E} \sum_{d \in D} (q_{ijd} \cdot Y_{ijd} \cdot X_{ij}) + (q_{jid} \cdot Y_{jid} \cdot X_{ij}) \right), \quad (3.8a)$$

$$\sum_{(s,j) \in A} \sum_{d \in D} Y_{sjd} - \sum_{(j,s) \in A} \sum_{d \in D} Y_{jds} = V, \quad (3.8b)$$

$$\sum_{(t,j) \in A} \sum_{d \in D} Y_{tjd} - \sum_{(j,t) \in A} \sum_{d \in D} Y_{jtd} = -V, \quad (3.8c)$$

$$\sum_{\substack{(i,n) \in A \\ i,n \neq s \\ i,n \neq t}} \sum_{d \in D} Y_{ind} - \sum_{\substack{(n,j) \in A \\ n,j \neq s \\ n,j \neq t}} \sum_{d \in D} Y_{njd} = 0, \quad (3.8d)$$

$$Y_{ijd} + Y_{jid} \leq u_{ijd} \cdot W_{ijd} \quad \forall (i, j) \in E, d \in D, \quad (3.8e)$$

$$\sum_{(i,j) \in E} r_{ij} \cdot X_{ij} \leq attack_budget, \quad (3.8f)$$

$$\sum_{\substack{d \in D: \\ d \neq d_0}} \sum_{(i,j) \in E} h_{ijd} \cdot W_{ijd} \leq defense_budget, \quad (3.8g)$$

$$\sum_{d \in D} W_{ijd} = 1 \quad \forall (i, j) \in E, \quad (3.8h)$$

$$Y_{ijd} \geq 0 \quad \forall (i, j) \in A, d \in D, \quad (3.8i)$$

$$X_{ij} \in \{0,1\} \quad \forall (i, j) \in E, \quad (3.8j)$$

$$W_{ijd} \in \{0,1\} \quad \forall (i, j) \in E, d \in D. \quad (3.8k)$$

The objective function (3.8a) expresses the three levels of optimization. The innermost maximization operator seeks to maximize the amount of flow on the network

after defenses and attacks are fixed. The middle operator has the objective of minimizing the amount of the resulting optimal flow by attacking edges for a given defense plan. The outer maximization operator chooses the optimal defense plan to maximize the worst-case flow over all attacks. Equation (3.8b) balances flow from the source node to the rest of the network. Equation (3.8c) balances the flow from the network to the terminal node. Equation (3.8d) represents balance of flow equations for all other nodes in the network that are not the source or destination nodes. Equation (3.8e) represent capacity constraints for each edge in the network, forcing total flow on an edge to be less than the maximum capacity for the edge under a defense plan and defense option chosen. Equation (3.8f) is a budget constraint on the attacker to ensure that the attacker can only attack as many edges as he or she can afford. Equation (3.8g) is a budget constraint on the defender which ensures the amount of network defense stays within a budget. Equation (3.8h) ensures each edge can only have one defense option chosen. Equations (3.8i) – (3.8k) enforce variable types.

For an example of nested defenses of the DAD problem in a maximum flow network, consider the network proposed by Alderson et al. (2013) in their appendix. There are six nodes in the network that have six parallel interconnecting directed arcs each. Node 1 is the source node and node 6 is the destination node. The arcs all have specific capacity limits. The network is shown in Figure 42. The undisturbed maximum flow of this network is limited to 36 units by the capacity of the directed arcs. The authors originally considered this network in an AD scenario in order to show that worst-case attacks to an undefended network may not be nested. We extend their analysis of this network to a DAD problem that includes defenses. Suppose the attacker has an attack budget of five attacks, and the defense budget is subject to parametric analysis between zero and five units. Defending an arc in the network costs one unit of defense budget, and attacking an arc costs one unit of attack budget. We do not allow for the addition of new edges in this network.

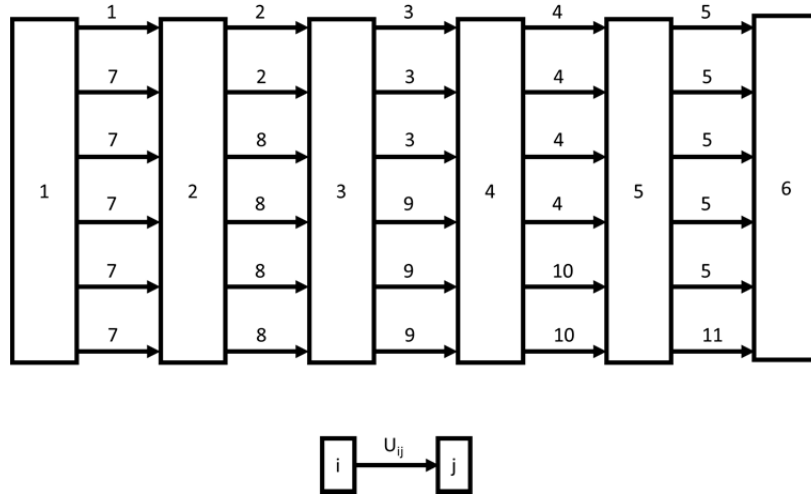


Figure 42. Maximum Flow Network. Adapted from Alderson et al. (2013)

The nested defense in this maximum flow problem would be formed as follows. If the defender has a budget of zero, the attacker will attack the five arcs from node 1. This worst-case attack will reduce the maximum flow of the network to one unit. If the defender has a budget of one unit, then one of the high capacity arcs from node 1 will be defended. The worst-case attack against this one defense will be the five highest capacity arcs from node 2. The maximum flow with one defense is two units. If the defender has a budget of two units, one high capacity arc from node 1 and one high capacity arc from node 2 will be defended. The worst-case attack against these two defenses would be the five highest capacity arcs from node 3. The maximum flow with two defenses would be three units. For example, the scenario of defense budget two units and attack budget five units is shown in Figure 43.

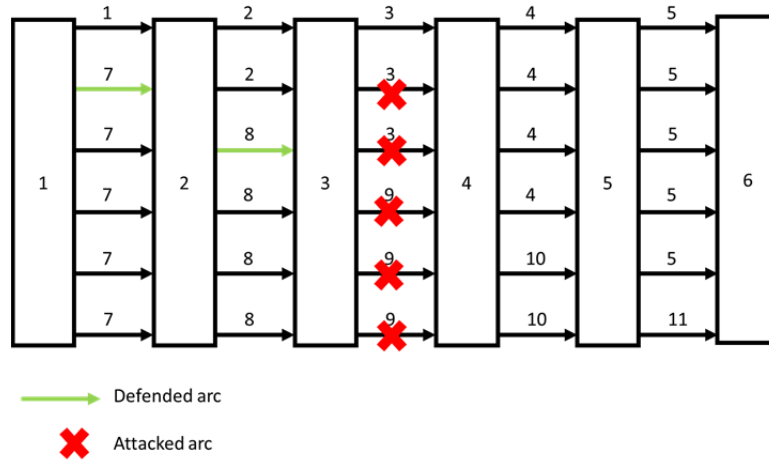


Figure 43. Maximum Flow Network with Defense Budget 2, Attack Budget 5

The remainder of the parameterization of the network defense budget follows a pattern. When the defender has a budget of three units, one of each of the high capacity arcs from nodes 1, 2, and 3 are defended. The worst-case attack against a defense budget of three units would attack five of the highest capacity arcs from node 4. The maximum flow with three defenses would be four units. When the defender has a budget of four units, one of each of the high capacity arcs from nodes 1, 2, 3, and 4 are defended. The worst-case attack against a defense budget of four units would be to attack five of the highest capacity arcs from node 5. The maximum flow with a defense budget of four units would be five. When the defender has a budget of five units, one of each of the high capacity arcs from nodes 1, 2, 3, 4 and 5 are defended. The worst-case attack against five defenses will be to attack the five undefended arcs from node 1. The maximum flow with five defenses would be seven units. The results of the optimal defenses and worst-case attacks for the parameterized defense budget are summarized in Table 28.

Table 28. Optimal Defenses and Attacks in Maximum Flow Network Example

Defense Budget	Arcs Defended	Arcs Attacked	Maximum Flow
0	None	5 highest capacity arcs (1, 2)	1
1	Arc (1, 2) with capacity 7	5 highest capacity arcs (2, 3)	2
2	Arc (1, 2) with capacity 7 Arc (2, 3) with capacity 8	5 highest capacity arcs (3, 4)	3
3	Arc (1, 2) with capacity 7 Arc (2, 3) with capacity 8 Arc (3, 4) with capacity 9	5 highest capacity arcs (4, 5)	4
4	Arc (1,2) with capacity 7 Arc (2, 3) with capacity 8 Arc (3, 4) with capacity 9 Arc (4, 5) with capacity 10	5 highest capacity arcs (5, 6)	5
5	Arc (1, 2) with capacity 7 Arc (2, 3) with capacity 8 Arc (3, 4) with capacity 9 Arc (4, 5) with capacity 10 Arc (5, 6) with capacity 11	5 undefended arcs (1, 2)	7

Table 28 shows that the defenses are nested because the defenses chosen for a defense budget of one unit are a subset of the defenses chosen for a defense budget of two units, and so on. We notice in this example maximum flow network that each nested defense systematically forms a *backbone* of highest capacity protected arcs for flow to travel from source node 1 to destination node 6. Each time the defense budget is increased, the new defense increases the resulting maximum flow of the network. The worst-case attack changes with each increase of the defense budget as it attempts to limit flow from source to destination as each new defense is added to the network.

Proposition 3–6: Increasing the defense budget in a DAD maximum flow network has a non-decreasing effect on the objective function.

Proof of proposition 3–6: DAD problems always assume that the attacker will make a worst-case attack. The addition of a defense in a DAD problem only occurs when a worst-case attack is changed to a new attack that has a smaller effect on the objective function value. By definition, no other attack can damage the objective function more than a worst-case attack. Therefore, the addition of a defense will result in a new worst-case attack that is no worse than the previous worst-case attack. The resulting objective

function value from the addition of a defense will be at least as good as the objective function value before the addition of the defense, or the defense will not be added to the network. In a DAD maximum flow problem, the addition of a defense will change a worst-case attack that results in a maximum flow that is at least as great or greater than the flow on the network before the defense was included.

I. NESTED DEFENSE SUMMARY

This chapter investigates the implications of producing a list of prioritized defenses for a system when the defense and/or attack budget is not known when defense decisions must be made. We use the term *nested defenses* to refer to a monotonic sequence of sets, where each set of defenses for a particular budget scenario contains the set of defenses for the next smaller budget scenario. It is clear in the literature that the best defenses chosen for a defense budget of two units is usually not a subset of the defenses chosen for a defense budget of three units, and so on.

We examine how nested defenses differs from the optimal set of defenses for any particular defense and attack budget scenario for the DAD minimum cost flow problem on the test network of Alderson et al. (2015). In particular, we quantify the *cost* of requiring a set of defenses to be monotonic. We examine various heuristic nesting strategies and compare their performance against known optimal solutions. Our heuristic algorithms choose an optimal defense for a particular budget scenario as an anchor point, and build nested defenses from it. The heuristic can either begin a nesting strategy from minimum, midrange or maximum budget values. We present three dimensional graphs to show the decision maker how an optimal defense differs from a heuristic nested defense in terms of system cost.

We present five different methods for the analyst to assess how far the nested defense solution is from the optimal solution when parametric programming analysis is required for unknown budgetary parameters. Vector norm and matrix norm methods are from linear algebra; optimality gap analysis is adapted from statistics; and minimax regret and equal likelihood analysis are from decision theory. In each method, the analyst is looking for the smallest possible measurement since the objective of the system is to

minimize cost. For the test network with uncertain budgets, the clear winner of nesting starting points is maximum defense and attack budgets. However, there is no guarantee that all of the measurement procedures contained will agree on the same starting point as closest to optimal. We define the best nested defense to be the set of defenses that is closest to the optimal non-nested solution over the set of unknown budget scenarios. Our example of suboptimal nesting demonstrates how our heuristic approaches may not produce the best set of nested defenses for a DAD minimum cost flow problem.

In order to find the best possible nested defense, we develop a new parametric programming formulation of the DAD minimum cost flow problem that accepts uncertainty in the defense or attack budgets. This new formulation works to minimize the overall system cost by simultaneously considering all unknown budget scenarios. Our formulation adds a block of constraints to enforce monotonicity of defenses across all budget scenarios. Our parametric programming formulation can find the nested defense that is closest to optimal based on three different measurement criteria. We demonstrate parametric programming formulations based on the 1-norm, squared 2-norm and infinity norm. We can conclude that our parametric programming formulation can generate the set of nested defenses that is as close as possible to the set of optimal non-nested solutions over all possible budget scenarios in terms of a vector norm measurement.

Lastly, we briefly describe nested defenses for the DAD maximum flow problem. We formulate a DAD maximum flow model. Solution procedures for the DAD maximum flow model are analogous to those presented for the DAD minimum cost flow problem. We use a test network of Alderson et al. (2013) to describe nested defenses on simple DAD maximum flow network. We conclude that adding additional defenses to a DAD maximum flow network will not degrade network performance.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. DAD CONSTRAINED SHORTEST PATH PROBLEM

A. PROBLEM DEFINITION

1. Introduction

The shortest path (SP) problem has been called “the simplest of all network flow problems” (Ahuja et al., 1993, p. 6). Fulkerson and Harding (1977) introduced the idea of attacking segments of a network to the SP problem by maximizing the length of a minimum shortest path. Golden (1978) furthered Fulkerson’s work and used the term *network interdiction* problem. Wood (1993) showed that the shortest path network interdiction problem is NP-complete. Improvements to the algorithm for solving the shortest path network interdiction problem were described by Israeli and Wood (2002).

The simple SP problem becomes the much more difficult constrained shortest path (CSP) problem when an additional budgetary side constraint is added, and the special network structure of the original problem can no longer be exploited (Ahuja et al., 1993, p. 599). Our research expands the complexity of the network interdiction problem by adding two new features. First, we expand the network interdiction problem to model constrained shortest paths. Second, we introduce the ability to defend segments of the network against attack. Even though the literature starts with and focuses on network flow problems, the algorithms in Alderson et al. (2011, 2014, & 2015) also apply to very general operator models, including those modeled as integer linear programming (ILP) problems. We picked CSP as the operator problem because it is a well understood specimen of an ILP that is *almost* a network flow problem. The addition of defenses to the network interdiction problem along with the addition of a side constraint creates the Defender-Attacker -Defender Constrained Shortest Path (DAD CSP) problem.

The concept behind the DAD CSP model is similar to the general DAD model presented in Chapter I. For DAD CSP, our objective is to travel from source to destination at a minimum cost, and the additional budgetary constraint specifies that the total time to travel from source to destination must be within a time budget. The attacker desires to interdict some of the arcs of the network graph, limited by an attack budget.

Each attack increases the cost of traversing an arc by a fixed amount. The defender has the ability to prevent attacks on a subset of arcs in the network, based on a given defense budget. We do not consider adding new arcs as defenses in this chapter. Thus, the DAD CSP problem seeks to choose defenses to minimize the cost of the constrained shorted path resulting from a cost maximizing worst-case attack.

We define the DAD CSP problem as follows. Given a network $G = (N, A)$ with a prescribed start node s , destination node t , arcs with costs c_{ij} and travel times t_{ij} , and an overall time budget, T , the classical CSP problem seeks the least cost path from s to t such that the time expended on the path is no larger than T . The choice of arcs on the constrained shortest path is governed by binary variables (Y). This is the simplest version of the *operator problem* (4.1a), which we seek to embellish. If we add a distance penalty, q_{ij} , to each arc so that the length of an attacked arc (X) becomes $c_{ij} + q_{ij}$, we obtain the *attacker problem* (4.1b). If we add the ability to protect a subset of arcs from an attack (W) constrained by a defense budget, we obtain the *defender problem* (4.1c). The constraint sets for defenses (Ψ), attacks (Ξ) and paths (Y) in (4.1a)-(4.1c) are similar to those in Chapter I, and are suppressed for clarity. The operator model constraint set in (4.1c) is dependent upon the values of defense variables, and is written as $Y(W)$.

$$\min_{Y \in Y} f(Y), \quad (4.1a)$$

$$\max_{X \in \Xi} \min_{Y \in Y} f(X, Y), \quad (4.1b)$$

$$\min_{W \in \Psi} \max_{X \in \Xi} \min_{Y \in Y(W)} f(W, X, Y). \quad (4.1c)$$

The feasible region ($Y(W)$) is the same as Chapter I, except that it contains a single, additional side constraint limiting the time duration of any feasible path to a time budget (T), shown in Equation (4.1d).

$$\sum_{(i,j) \in A} \sum_{d \in D} t_{ij} \cdot Y_{ijd} \leq T. \quad (4.1d)$$

The side constraint rescinds the network structure of the problem, and the path variables (Y) must then be treated explicitly as binary variables. Consequently, the

constraint set for the inner operator model ($Y(W)$) does not allow for the direct substitution of continuous variables for the binary path decision variables (Y) as seen in Chapter 1. The formulation in (4.1c) is extremely difficult to solve.

In order to relieve the difficulty in (4.1c), we begin by isolating the operator model by fixing the values of the defense variables (W) and attack variables (X) with the use of the hat notation (^) in (4.1e).

$$\min_{Y \in Y(\hat{W})} f(\hat{W}, \hat{X}, Y). \quad (4.1e)$$

Ahuja et al. (1993, pp. 599–605) shows that Lagrangian relaxation of a side constraint can be used to regain the network structure of a CSP problem. The resulting heuristic algorithm can be converted to a competitive optimal algorithm as shown in Carlyle, Royset and Wood (2008).

We relax some of the constraints of the operator model ($Y(W)$) by incorporating them into a modified objective function with a Lagrange multiplier (μ) that penalizes infeasibility. We use the tilde notation (\sim) to signify the modification of the objective function (f) and operator model constraint set ($Y(W)$) in Equation (4.1f). Specifically, we use $\tilde{Y}(\hat{W})$ to represent the feasible region of a shortest path without the side constraint. We use \tilde{f} to represent the objective function with the penalized side constraint.

$$L(\hat{W}, \hat{X}, \mu) = \min_{Y \in \tilde{Y}(\hat{W})} \tilde{f}(\hat{W}, \hat{X}, Y, \mu) = \min_{Y \in \tilde{Y}(\hat{W})} f(\hat{W}, \hat{X}, Y) + \mu \cdot \left[\sum_{(i,j) \in A} \sum_{d \in D} t_{ij} \cdot Y_{ijd} - T \right]. \quad (4.1f)$$

We can solve the Lagrangian multiplier problem by choosing the value of the multiplier that maximizes the Lagrangian multiplier function (L), as shown in (4.1g) (Ahuja et al., 1993, p. 608).

$$\max_{\mu \geq 0} L(\hat{W}, \hat{X}, \mu). \quad (4.1g)$$

The maximization of the Lagrange multiplier function (L) can be accomplished various ways. One way is to iteratively update the value of the Lagrange multiplier (μ) based on a subgradient calculation for a fixed defense (W) and attack (X). This is frequently coupled with a trust region method to force termination. See Ahuja et al. (1993, pp. 608–615) for a complete discussion. Since bounds of Lagrangian relaxation do not always converge, we employ a path enumeration technique of Carlyle, Royset and Wood (2008) to close the optimality gap.

We incorporate the Lagrange multiplier function maximization of (4.1g) into the tri-level model to see the effect of the relaxation on the DAD CSP problem in (4.1h).

$$\min_{W \in \Psi} \max_{X \in \Xi} \left[\max_{\mu \geq 0} \min_{Y \in \Upsilon(W)} \tilde{f}(W, X, Y, \mu) \right]. \quad (4.1h)$$

An alternative method to maximize the Lagrange multiplier function (L) is to focus on the bi-level attacker-defender (AD) model. For the AD version of CSP where defenses (W) are fixed, we take advantage of the side by side relationship of the two maximization operators in (4.1h). We can characterize the Lagrange multiplier (μ) as a decision variable, and combine the maximization operators. We define the bi-level AD CSP Lagrange variable problem in (4.1i) to use the Lagrange multiplier (μ) as a decision variable. Essentially, the attacker gets to choose the arcs to attack (X) and the penalties (μ) that the operator incurs for infeasibility of the side constraint.

$$\max_{\substack{X \in \Xi, \\ \mu, \mu \geq 0}} \min_{Y \in \Upsilon(\hat{W})} \tilde{f}(\hat{W}, X, Y, \mu). \quad (4.1i)$$

Since the relaxation of the side constraint allows us to treat the path choice variables (Y) as continuous variables, we can construct the dual formulation of the innermost minimization operator from (4.1i). The Lagrangian relaxation of bi-level AD CSP can be rewritten as a bi-level optimization model in (4.1j) by defining dual variables to represent the nodes (π) and arcs (α) of the network. We define (4.1j) as the *Lagrange Variable dual ILP problem*.

$$\max_{\substack{X, \pi, \alpha \in \Xi, \\ \mu: \mu \geq 0}} \tilde{f}(\hat{W}, X, \pi, \alpha, \mu). \quad (4.1j)$$

We can incorporate the Lagrange variable dual ILP problem into the tri-level model to see the effect of the relaxation on the DAD CSP problem in (4.1k).

$$\min_{W \in \Psi} \max_{\substack{X, \pi, \alpha \in \Xi, \\ \mu: \mu \geq 0}} \tilde{f}(W, X, \pi, \alpha, \mu). \quad (4.1k)$$

Expression (4.1k) is a relaxation of the original DAD CSP problem of (4.1c). Therefore, (4.1k) is not guaranteed to produce optimal solutions. Decomposition of (4.1k) over a series of iterations produces an AD sub problem in (4.1l) where the defense (W) is held constant. Decomposition produces a DAD master problem in (4.1m) where the attack (X), dual (α, π) and Lagrange (μ) variables are held constant.

$$\max_{\substack{X, \pi, \alpha \in \Xi, \\ \mu: \mu \geq 0}} \tilde{f}(\hat{W}, X, \pi, \alpha, \mu), \quad (4.1l)$$

$$\min_{W \in \Psi} \tilde{f}(W, \hat{X}, \hat{\pi}, \hat{\alpha}, \hat{\mu}). \quad (4.1m)$$

Decomposition produces valid lower bounds on the original DAD CSP problem instance because it incorporates Lagrangian relaxation. The decomposition in (4.1l) and (4.1m) produce heuristic solutions to DAD CSP instances more quickly than (4.1c) or (4.1h). We develop algorithms to incorporate the efficiencies within (4.1k) to produce optimal or provably near optimal solutions to DAD CSP instances more quickly than traditional nested decomposition techniques applied to (4.1c).

Proposition 4–1. DAD CSP is NP-Hard.

Proof of Proposition 4–1: When defense and attack budgets are restricted to zero units, DAD CSP reduces to the decision version of CSP, which is NP-Complete (Ahuja et al., 1993, p. 798). Therefore, DAD CSP is NP-Hard.

Note that if the defense budget is restricted to zero units and the side constraint time budget is not binding, then DAD CSP reduces to the network interdiction SP problem, the decision version of which is also NP-Complete (Wood, 1993). Evaluating a

feasible defense requires solving an NP-Complete AD problem. Finding the optimal defense adds an additional layer of complexity on top of an NP-Complete problem. It is unlikely that DAD CSP is even in NP.

Over the last few years, algorithms for solving instances of DAD SP have appeared in the literature. Alderson et al. (2011) gives a higher-level explanation for decomposition techniques for the tri-level optimization model for DAD SP. Alderson et al. (2014) provides a tutorial on the mathematical details for solving the tri-level DAD SP optimization model with decomposition procedures. To the best of our knowledge, the existing literature does not address the tri-level DAD CSP optimization problem, nor any solution procedures specifically developed for it.

The intent of this chapter is to investigate the application of Lagrangian relaxation to the DAD model for the CSP problem. We apply the nested decomposition methods found in Alderson et al. (2014 and 2015) to solve DAD CSP problem instances on small to medium sized test networks. We develop new heuristic methods to quickly find an estimated solution to DAD CSP instance using Lagrangian relaxation of the side constraint. We combine the heuristic methods with nested decomposition to obtain provably optimal solutions faster than using nested decomposition by itself. We apply our methods to a real-world problem to demonstrate scalability to large networks.

2. Problem Formulation

We define the sets, data, decision variables and the formulation of the DAD CSP problem in system of Equations (4.2), using a structure similar to Alderson et al. (2014).

Defender-Attacker-Defender Constrained Shortest Path (DAD CSP) Problem:

Sets:

$n \in N$	Nodes of the network. (Note: i and j are used as aliases for n) (Note: node s is start of network; node t is end of network)
$(i, j) \in A$	Arcs of the network
$d \in D$	Defense choices for an arc. (Note: d_0 is an undefended arc)
$k \in K$	Iteration of inner decomposition
$k' \in K'$	Iteration of outer decomposition

Data:

c_{ij}	Cost to travel on arc (i, j).
h_{jd}	Cost to defend arc (i, j) with defense d.
q_{ijd}	Penalty cost to travel on attacked arc (i, j) with defense d.
r_{ij}	Cost to attack undirected edge (i, j).
t_j	Time required to traverse arc (i, j).
$time_budget$	Maximum time allowed to traverse a path from node s to node t.
$attack_budget$	Maximum budget for attacks on the network.
$defense_budget$	Maximum budget for defenses for the network.

Decision Variables:

W_{ijd}	1 if defense d is chosen for arc (i, j) , 0 otherwise.
X_{ij}	1 if arc (i, j) is attacked, 0 otherwise.
Y_{jd}	1 if arc (i, j) with defense d is chosen for shortest path, 0 otherwise.
$Y_{ijk'}$	1 if arc (i, j) with defense d is chosen for path in outer iteration k' .
Z_{AD}	Objective function value of AD inner master problem
Z_{DAD}	Objective function value of DAD Outer Master Problem

Additional Fixed Values of Decision Variables Used in Decomposition:

\hat{X}_{ij}	1 when arc (i, j) is attacked in inner decomposition.
$\hat{X}_{ijk'}$	1 when arc (i, j) is attacked in outer decomposition iteration k' .
\hat{W}_{ijd}	1 when arc (i, j) is assigned defense d in outer decomposition.
\hat{Y}_{ijk}	1 when arc (i, j) with defense d is chosen for path in iteration k .

DAD CSP Formulation:

$$\min_{W_{ijd}} \max_{X_{ij}} \min_{Y_{jd}} \sum_{(i,j) \in A} \sum_{d \in D} (c_{ij} + q_{ijd} \cdot X_{ij}) \cdot Y_{jd}, \quad (4.2a)$$

Subject to:

$$\sum_{j:(i,j) \in A} \sum_{d \in D} Y_{jd} - \sum_{j:(j,i) \in A} \sum_{d \in D} Y_{jd} = \begin{cases} -1 & \text{if } i = s \\ 0 & \forall i \in N \setminus \{s, t\} \\ +1 & \text{if } i = t \end{cases}, \quad (4.2b)$$

$$Y_{ijd} \leq W_{ijd} \quad \forall (i, j) \in A, d \in D, \quad (4.2c)$$

$$\sum_{(i,j) \in A} r_{ij} \cdot X_{ij} \leq \text{attack_budget}, \quad (4.2d)$$

$$\sum_{(i,j) \in A} \sum_{\substack{d \in D: \\ d \neq d_0}} h_{ijd} \cdot W_{ijd} \leq \text{defense_budget}, \quad (4.2e)$$

$$\sum_{d \in D} W_{ijd} = 1 \quad \forall (i, j) \in A, \quad (4.2f)$$

$$\sum_{(i,j) \in A} \sum_{d \in D} t_{ij} \cdot Y_{ijd} \leq \text{time_budget}, \quad (4.2g)$$

$$Y_{ijd} \in \{0, 1\} \quad \forall (i, j) \in A, d \in D, \quad (4.2h)$$

$$X_{ij} \in \{0, 1\} \quad \forall (i, j) \in A, \quad (4.2i)$$

$$W_{ijd} \in \{0, 1\} \quad \forall (i, j) \in A, d \in D. \quad (4.2j)$$

The objective function (4.2a) expresses the three levels of optimization. The innermost minimization operator seeks to minimize the cost of a path through the network after defenses and attacks are fixed. The middle maximization operator has the objective of maximizing the cost of the resulting path through the network, by attacking arcs for a given defense plan. The outer minimization operator chooses the optimal defense plan to minimize the worst-case cost over all attacks. Equation (4.2b) represents balance of flow equations for each node in the network, where one unit departs the start node (s), and one unit arrives at the terminal node (t). Equation (4.2c) only allows an edge to be considered for the path with a defense option chosen. Equation (4.2d) is a budget constraint on the attacker to ensure that the attacker can only attack as many edges as he or she can afford. Equation (4.2e) is a budget constraint on the defender which ensures the amount of network defense stays within a budget. Equation (4.2f) ensures each edge of the network can only have one defense option chosen. Equation (4.2g) is the “side constraint” that changes the SP problem to a CSP problem. Equation (4.2g) ensures the path chosen must have a total time less than the overall time budget. Equation (4.2g) destroys the special network structure of the DAD problem that had been exploited in previous chapters. Because of the side constraint, Equation (4.2h) must restrict the decision variable for path segments to binary values, which is the reason that the CSP problem is much more difficult than the SP problem. The binary decision variable

restriction present in Equation (4.2h) can be relaxed in the SP problem, but it cannot be relaxed in the CSP problem. Equations (4.2i) and (4.2j) enforce binary restrictions.

At a minimum, we have two possible defense choices for each arc. Defense choices in our application either leave an arc undefended (d_0) or choose to defend an arc (d_1). Our development of DAD CSP is based on two defense choices. However, our formulation can support more than two defense choices. Note that we have to choose some defense for every arc, which includes no defense (d_0). In this formulation, it is technically possible to attack an arc that has been defended. But, we choose the penalty (q) for traversing a defended arc (d_1) to be zero units, which makes it unattractive to attack a defended arc.

3. Regular Nested Decomposition Approach

The only method in the literature available for solving CSP DAD in system of Equations (4.2) is to perform a nested decomposition that is somewhat similar to the Benders (1962) algorithm (Alderson et al. 2011 & 2014). The significant difference between our decomposition and Benders (1962) decomposition is that our sub problem contains discrete variables. Thus, our nested decomposition must make the repeated use of solution elimination constraints (SEC) to prevent cycling. For a discussion of anti-cycling in these algorithms, see Alderson et al. (2014). The nested decomposition of CSP DAD is detailed in system of Equations (4.3). The nested decomposition features an inner and outer decomposition, where the inner decomposition addresses the inner minimization of path variables (Y) and middle maximization of attack variables (X), and the outer decomposition takes the result of the inner decomposition and addresses the outer minimization of defense variables (W). The nested decomposition requires the introduction of new sets that govern the iterations of both inner and outer decomposition, and new variables representing paths for each iteration of outer decomposition. We use a hat (^) on a variable to represent a fixed value that is referenced during decomposition. We refer to this nested decomposition technique as *regular* nested decomposition in order to distinguish it from other methods that we will introduce later in the chapter.

The inner decomposition AD sub problem takes a fixed defense (\hat{W}), a fixed attack (\hat{X}) and solves for the minimum cost path (Y) through the network.

a. CSP Inner Decomposition AD Subproblem Formulation:

$$\min_{Y_{ijd}} \sum_{(i,j) \in A} \sum_{d \in D} (c_{ij} + q_{ijd} \cdot \hat{X}_{ij}) \cdot Y_{ijd}, \quad (4.3a)$$

Subject to:

$$\sum_{j:(i,j) \in A} \sum_{d \in D} Y_{ijd} - \sum_{j:(j,i) \in A} \sum_{d \in D} Y_{jid} = \begin{cases} -1 & \text{if } i = s \\ 0 & \forall i \in N \setminus s, t \\ +1 & \text{if } i = t \end{cases}, \quad (4.3b)$$

$$\sum_{(i,j) \in A} Y_{ijd} \leq \hat{W}_{ijd} \quad \forall (i,j) \in A, d \in D, \quad (4.3c)$$

$$\sum_{(i,j) \in A} \sum_{d \in D} t_{ij} \cdot Y_{ijd} \leq \text{time_budget}, \quad (4.3d)$$

$$\sum_{\substack{(i,j) \in A: d \in D \\ \hat{Y}_{ijk}=0}} Y_{ijd} + \sum_{\substack{(i,j) \in A: d \in D \\ \hat{Y}_{ijk}=1}} (1 - Y_{ijd}) \geq 1 \quad \forall k \in K, \quad (4.3e)$$

$$Y_{ijd} \in \{0,1\} \quad \forall (i,j) \in A, d \in D. \quad (4.3f)$$

The objective function (4.3a) minimizes the cost of a shortest path based on fixed attacks to the network. Equation (4.3b) restates Equation (4.2b). Equation (4.3c) resembles Equation (4.2c), except the defense variables are now fixed values. Equation (4.3d) restates Equation (4.2g). Equation (4.3e) is a SEC to prevent cycling in the inner decomposition. Equation (4.3e) is a SEC that is only activated in inner decomposition iterations when a choice of path variables (Y) is repeated from a previous iteration. Equation (4.3f) restates (4.2h).

The inner decomposition AD master problem takes a fixed defense (\hat{W}) and a list of fixed paths (\hat{Y}) for each iteration (K) to solve for the worst-case attack (X) that maximizes the length of the shortest path. During each iteration (K) of inner decomposition, the AD sub problem takes a set of fixed defense variables (\hat{W}) and fixed attack variables (\hat{X}) and determines a shortest path through the network. The choice of shortest path variables (Y) for the current iteration (K) become an input to the AD master

problem. The cost associated with the shortest path updates the inner decomposition lower bound if it is the highest cost of all iterations.

b. CSP Inner Decomposition AD Master Problem Formulation:

$$\max_{Z_{AD}, X_{ij}} Z_{AD}, \quad (4.3g)$$

Subject to:

$$Z_{AD} \leq \sum_{(i,j) \in A} \sum_{d \in D} (c_{ij} + q_{ijd} \cdot X_{ij}) \cdot \hat{Y}_{ijdk} \quad \forall k \in K, \quad (4.3h)$$

$$\sum_{(i,j) \in A} r_{ij} \cdot X_{ij} \leq \text{attack_budget}, \quad (4.3i)$$

$$\sum_{\substack{(i,j) \in A: \\ \hat{X}_{ijk}=0}} X_{ij} + \sum_{\substack{(i,j) \in A: \\ \hat{X}_{ijk}=1}} (1 - X_{ij}) \geq 1 \quad \forall k' \in K, \quad (4.3j)$$

$$X_{ij} \in \{0,1\} \quad \forall (i,j) \in A. \quad (4.3k)$$

The objective function (4.3g) and (4.3h) work together to form the inner master problem decomposition to maximize the cost of fixed shortest paths by the choice of attack variables (X). Equation (4.3i) restates Equation (4.2d). Equation (4.3j) is a SEC to prevent cycling in the outer decomposition. Equation (4.3j) is only activated in outer decomposition iterations when a choice of attack variables (X) is repeated from a previous iteration of outer decomposition. Equation (4.3k) restates Equation (4.2i).

The AD master problem next determines a choice of attack variables (X) to maximize the cost of the shortest path. Each iteration of the AD master problem represents an opportunity to update the inner decomposition upper bound. When the inner decomposition lower bound is updated, the attack associated with that iteration is saved as the best known attack. We refer to this set of best attack variable values as X_{star} (X^*). The best attack result is then input to the DAD outer decomposition master problem. The cost of the best attack updates the upper bound of the outer decomposition.

The outer decomposition master problem takes a list of fixed attacks over a series of outer iterations ($\hat{X}_{k'}$) and solves for the best defense (\hat{W}) and the resulting path for each outer iteration ($Y_{k'}$) to minimize the shortest path for the original problem instance.

c. Outer Decomposition DAD Master Problem Formulation:

$$\min_{Z_{DAD}, W_{ijk'}, Y_{ijk'}} Z_{DAD}, \quad (4.3l)$$

Subject to:

$$Z_{DAD} \geq \sum_{(i,j) \in A} \sum_{d \in D} (c_{ijd} + q_{ijd} \cdot \hat{X}_{ijk'}) \cdot Y_{ijk'}, \quad \forall k' \in K, \quad (4.3m)$$

$$\sum_{j:(i,j) \in A} \sum_{d \in D} Y_{ijk'} - \sum_{j:(j,i) \in A} \sum_{d \in D} Y_{jik'} = \begin{cases} -1 & \text{if } i = s \\ 0 & \forall i \in N \setminus s, t \\ +1 & \text{if } i = t \end{cases}, \quad (4.3n)$$

$$\sum_{(i,j) \in A} Y_{ijk'} \leq W_{ijd} \quad \forall (i,j) \in A, d \in D, k' \in K, \quad (4.3o)$$

$$\sum_{d \in D} W_{ijd} = 1 \quad \forall (i,j) \in A, \quad (4.3p)$$

$$\sum_{(i,j) \in A} \sum_{d \in D} t_{ij} \cdot Y_{ijk'} \leq \text{time_budget} \quad \forall k' \in K, \quad (4.3q)$$

$$Y_{ijk'} \in \{0,1\} \quad \forall (i,j) \in A, d \in D, k' \in K, \quad (4.3r)$$

$$W_{ijd} \in \{0,1\} \quad \forall (i,j) \in A, d \in D. \quad (4.3s)$$

The objective function (4.3l) and (4.3m) work together to form the outer decomposition master problem to minimize the cost of shortest paths with fixed attacks by the choice of defense and path variables. Equation (4.3n) and (4.3o) are similar to Equations (4.2b) and (4.2c), except that iteration subscripts “ k ” have been added to the path variables for each round of outer decomposition. Equation (4.3p) restates Equation (4.2f). Equations (4.3q) and (4.3r) are similar to Equations (4.2g) and (4.2i), respectively, but the addition of subscript “ k ” allows for different path variables for each iteration of outer decomposition. Equation (4.3s) restates Equation (4.2j).

The outer decomposition works through a set of iterations (K') until a stopping criterion is reached. For each outer decomposition iteration, the best attack (X) from the inner decomposition is used as a fixed input to update an outer lower bound. The outer decomposition then determines a choice of defense variables (W) to create a shortest path. The shortest path variables (Y) are dependent upon a fixed attack (X) for each outer iteration (K'). The outer decomposition lower bound may be updated every time defense variables (W) find a shorter (lower cost) path. When the outer decomposition lower

bound changes, the values of the defense variables used in that iteration are saved as the best defense. We refer to the best defense as *W star* (W^*). The outer decomposition continues until the outer upper and lower bounds match. The result of the outer decomposition is the choice of defenses (W).

The algorithm for implementing regular nested decomposition of DAD CSP is similar to the DAD minimum cost flow problem presented in Chapter I. The only significant algorithmic difference between regular nested decomposition of DAD CSP vs. DAD minimum cost flow that of Chapter I is that the operator model decision variables (Y) are now restricted to binary values. This difference means that SEC that were present in outer decomposition must now also appear in the inner decomposition to prevent cycling. We present the extra SEC in (4.3e). The algorithm of Chapter I can be adapted *mutatis mutandis* for the decomposition of the DAD CSP problem in system of Equations (4.3). Regular nested decomposition of system of Equations (4.3) can solve small to medium sized instances of DAD CSP optimally. Large-scale DAD CSP problem instances may struggle to find optimal solutions in a reasonable amount of time.

The number of binary variables in a DAD CSP problem instance can be quite large, since the binary decision variables for path (Y), attack (X) and defense (W) apply to every arc in the network. The total number of binary variables in a problem is three times the cardinality of the arc set. If we let “ m ” represent the number of arcs in the network, then the total number of binary variables in the DAD CSP problem is $3m$, as shown in Equation (4.3t). Later sections of the chapter are devoted to finding methods that reduce the number of binary variables in the DAD CSP problem by taking into account the structure of the side constraint.

DAD CSP total binary decision variables =

$$m \text{ defense variables} + m \text{ attack variables} + m \text{ path choice variables} = 3m \quad (4.3t)$$

B. ARTIFICIAL TEST NETWORKS

We created two networks for testing the DAD CSP problem. The first is a small instance derived from Ahuja et al. (1993) for testing purposes and to understand the

sequence of solutions examined by our algorithms. The second test instance is a medium-sized mesh-like network with random data to test the limits of computation.

The first test network is a small six node textbook example shown in Figure 44 (Ahuja et al., 1993, p. 599). Each directed arc has two associated data values. The first value is the cost for traveling on the arc and the second value is the time required to traverse the arc. This network was chosen because it has been studied in the literature and for characteristics that will be used in a later section. The starting node is node 1 and the terminal node is node 6. The shortest path in the network is to traverse the arcs (1, 2), (2, 4), (4, 6), with a cost of three units and usage of 18 time units. In order to make the side constraint meaningful, we must choose a time budget less than 18 units. A time budget of 14 units is used with this network, which makes the original shortest path infeasible.

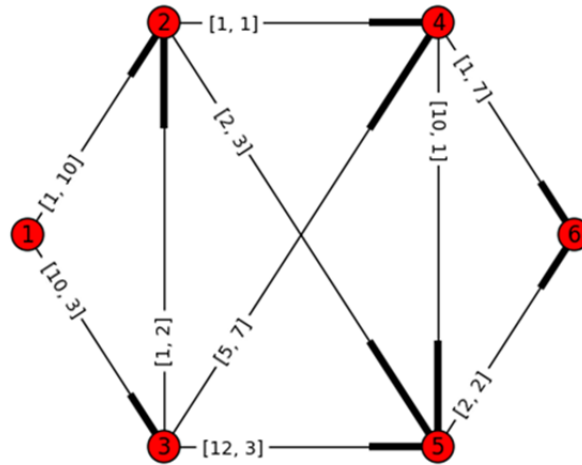


Figure 44. Six Node Test Network. Adapted from Ahuja et al. (1993, p. 599)

The second test network is a medium size grid network based on the CSP problem of Royset, Carlyle and Wood (2009). It is a grid with 50 nodes and 124 directed arcs, shown in Figure 45. This network is acyclic since each of the directed arcs points from left to right. The values of the arc cost and time are random integers uniformly distributed between one and ten, inclusive. The random values are obtained from the random integer generator found in Python programming software version 2.7 with a seed value of one. The arc costs and time values are not shown in Figure 45 for the sake of clarity. The

randomly assigned cost and time values for the arcs for the 50 node grid network are in Table 29. All of our tests are based on the costs and times in Table 29.

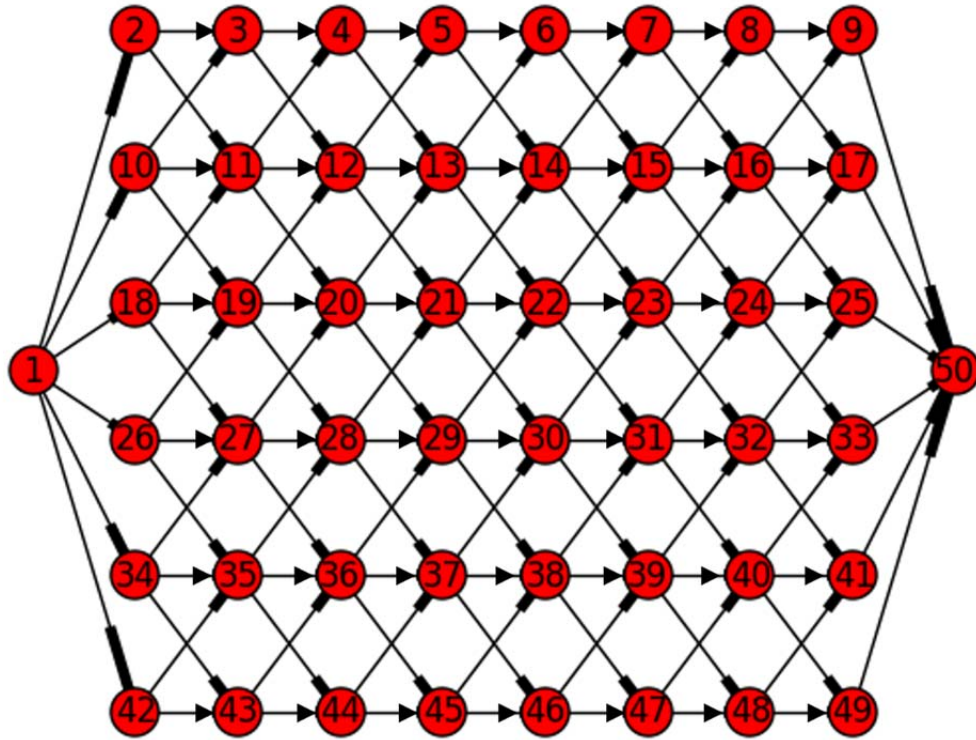


Figure 45. 50 Node Test Network

Table 29. Arc Cost and Time Values for 50 Node Test Network

(i, j)	c_{ij}	t_{ij}	(i, j)	c_{ij}	t_{ij}	(i, j)	c_{ij}	t_{ij}	(i, j)	c_{ij}	t_{ij}
1,2	2	9	14,23	8	7	1,26	1	1	36,37	9	10
2,3	8	3	15,8	4	5	26,19	8	3	36,45	8	6
2,11	5	5	15,16	6	8	26,27	2	7	37,30	4	4
3,4	7	8	15,24	6	4	26,35	4	1	37,38	3	7
3,12	1	1	16,9	5	1	27,20	2	6	37,46	5	2
4,5	9	5	16,17	1	8	27,28	2	3	38,31	2	7
4,13	8	1	16,25	10	6	27,36	8	5	38,39	3	5
5,6	5	8	17,50	4	2	28,21	4	5	38,47	4	9
5,14	3	10	1,18	6	10	28,29	1	4	39,32	9	1
6,7	10	1	18,11	8	6	28,37	5	2	39,40	3	4
6,15	1	6	18,19	9	3	29,22	2	9	39,48	10	8
7,8	10	4	18,27	6	10	29,30	6	3	40,33	4	3
7,16	3	5	19,12	6	5	29,38	7	9	40,41	7	9
8,9	1	3	19,20	3	6	30,23	1	1	40,49	10	4
8,17	5	5	19,28	10	1	30,31	2	8	41,50	9	7
9,50	3	3	20,13	8	9	30,39	2	8	1,42	5	10
1,10	3	5	20,21	9	8	31,24	7	6	42,35	3	8
10,3	3	1	20,29	9	6	31,32	3	10	42,43	1	2
10,11	9	6	21,14	6	5	31,40	8	6	43,36	10	3
10,19	7	2	21,22	1	9	32,25	3	7	43,44	8	7
11,4	10	9	21,30	6	2	32,33	4	6	44,37	9	4
11,12	2	4	22,15	6	5	32,41	4	7	44,45	4	3
11,20	8	8	22,23	4	4	33,50	1	3	45,38	9	7
12,5	10	5	22,31	6	7	1,34	10	9	45,46	10	9
12,13	9	7	23,16	7	5	34,27	4	9	46,39	2	6
12,21	4	6	23,24	1	3	34,35	4	10	46,47	2	1
13,6	9	9	23,32	2	6	34,43	8	5	47,40	1	9
13,14	6	6	24,17	9	8	35,28	3	1	47,48	8	9
13,22	1	3	24,25	8	9	35,36	9	1	48,41	4	7
14,7	8	5	24,33	3	9	35,44	9	10	48,49	8	4
14,15	2	6	25,50	7	1	36,29	6	2	49,50	6	3

The shortest cost path in this network has a cost of 17 distance units and a usage of 43 time units. Conversely, the shortest time path in this network uses 18 time units with cost of 42 distance units. Note that the shortest path cost of 17 distance units is much different than the quickest path cost of 42 distance units. Figure 46 plots of the shortest cost path with a solid pink line and the shortest time path with a brown dashed line for the

50 node grid test network. Our goal for testing DAD CSP with the 50 node test network will be to minimize cost with a time budget as the side constraint. In our tests, we minimize cost and use a side constraint time budget of 40 units with this network so that the side constraint is meaningful. Note that a time budget of 40 time units renders the unconstrained shortest cost path infeasible because that path requires 43 time units. There are many non-shortest paths with total time values below 40 units. We choose 40 units because a reasonable number of feasible paths may be considered by the defender and attacker, and it also forces the side constraint to be binding.

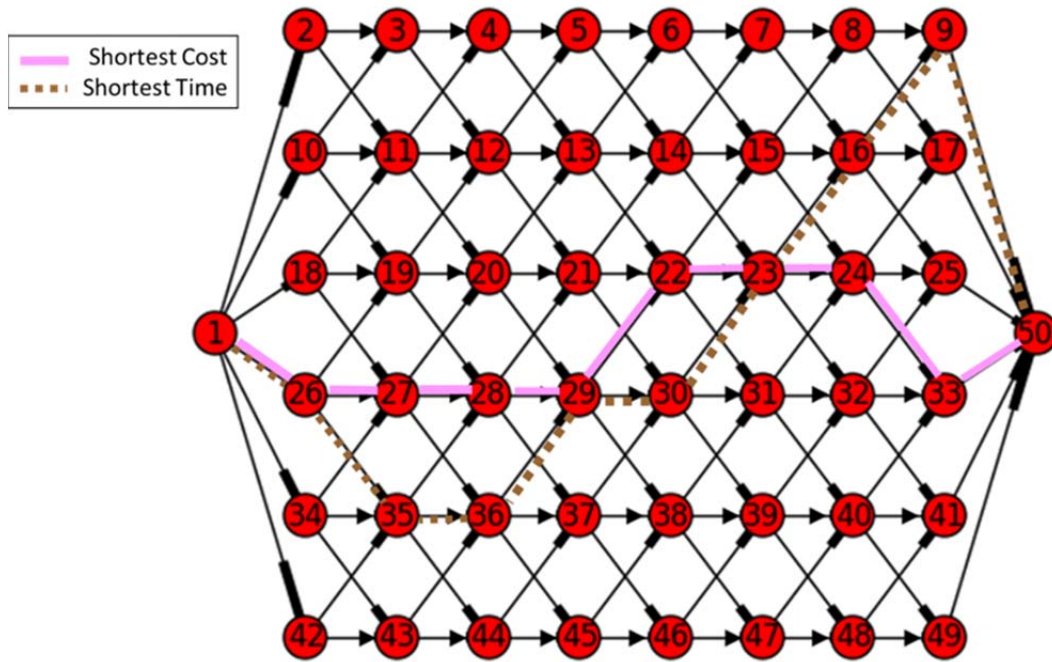


Figure 46. Shortest Cost and Shortest Time Paths for 50 Node Test Network

Our tests with these networks were conducted on Windows 7 computers with dual Xeon E-7 18 core processors operating at 2.3 GHz, with 128 GB of RAM and solid state hard drives. We use the General Algebraic Modeling System (GAMS) modeling software version 24.5 (GAMS, 2016) with the Industrial Business Machines (IBM) CPLEX solver version 12.6 (IBM, 2016). The algorithmic decomposition improvements described in later sections of this chapter are implemented with Python programming language version 2.7 interacting with the GAMS optimization software.

For both test networks we impose a penalty cost (q) of 25 units for traveling on an undefended (d_0) and attacked arc in the shortest path. A penalty of 25 units represents 2.5 times the maximum cost of any arc in the 50 node test network. The penalty cost (q) is zero units for traveling on undefended arcs that are not attacked. Furthermore, the penalty cost (q) is zero units for traveling on a defended arc, regardless of whether it is attacked or not. Zero penalty (q) for travel on a defended arc (d_1) eliminates the incentive for an attack to occur on a defended arc. We choose the cost of attacking any arc to be one unit of attack budget. We choose the cost of defending any arc to be one unit of defense budget. We set the tolerances for inner and outer decompositions to be less than one unit, which forces our algorithms to converge on the optimal solution. We revisit setting solution tolerances later in the chapter when we test a large real world network, for which optimal solutions are not practical.

The regular nested decomposition in system of Equations (4.3) can solve instances of DAD CSP for both test networks. We observe from both test networks that the solution time for “regular” nested decomposition of DAD CSP increases as the size of the defense and attack budgets increase. The size of the network in terms of the number of nodes and arcs also plays a significant role, with larger networks taking longer time to solve than smaller networks. The 50 node grid test network shows examples where the time to obtain a solution to the DAD CSP problem becomes longer as the budget scenario increases. Figure 47 plots the amount of solution time required in nested decomposition for various defense and attack budget scenarios on the 50 node grid test network.

DAD CSP Nested Decomposition Solution Times

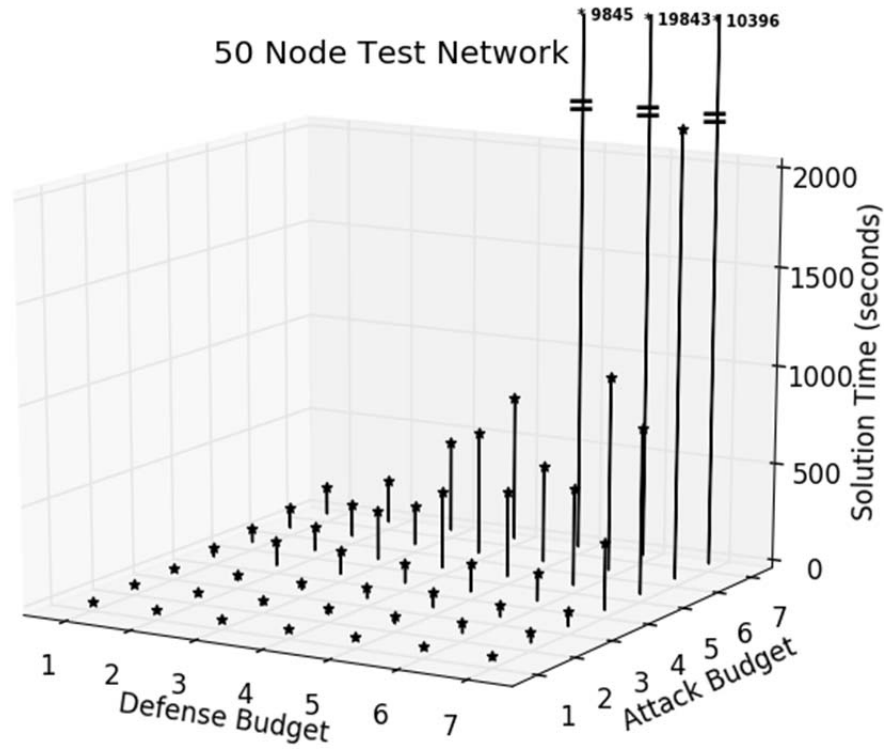


Figure 47. Nested Decomposition Solution Times in 50 Node Test Network

We can see clear evidence in Figure 47 that the nested decomposition solution begins to take a larger amount of time as the defense and attack budgets increase. With the 50 node grid test network, problem instances with an attack budget of seven units have a dramatic increase in solution time than instances with an attack budget of six units. A similar, but less dramatic, effect occurs when changing the problem instance attack budget from six to five units. Smaller attack budgets of four units or less appear to have a negligible effect on the time to obtain a solution. Additionally, increasing the defense and attack budgets to eight units appears to be a tipping point in the ability for our regular nested decomposition implementation in GAMS optimization software to find a solution to DAD CSP. We found that the DAD CSP problem instance for the 50 node test network with defense and attack budgets of eight units can take over 24 hours to compute without convergence on an optimal solution in regular nested decomposition.

This chapter explores methods to either estimate or solve DAD CSP problem instances faster than the regular nested decomposition technique. The challenge is to find an algorithm improvement to regular nested decomposition that can speed up the amount of time to get an answer to a DAD CSP problem instance.

C. LAGRANGIAN RELAXATION OF OPERATOR MODEL

In this section, we seek to transform the formulation of DAD CSP to reduce the number of binary decision variables in the problem. The reduction in the number of binary decision variables may improve the ability of the problem to solve more quickly. We focus on the operator model, and relax the path choice variables (Y) to continuous variables through the technique of Lagrangian relaxation.

Lagrangian relaxation can be used to transform a difficult integer or binary programming problem into a simpler model by relaxing the difficult constraints and adding a term in the objective to penalize infeasibility (Fisher, 1981). The DAD CSP problem is a difficult binary program to solve because the side constraint time budget eliminates the special network structure of the problem. Fisher states that moving the side constraint into the objective function “produces a Lagrangian problem that is easy to solve and whose optimal value is a lower bound (for minimization problems) on the optimal value of the original problem” (1981, p. 1). We use the technique of Lagrangian relaxation described in Ahuja et al. (1993, pp. 599–614) to move the side constraint time budget of Equation (4.2g) into the objective function. We introduce a Lagrange multiplier (μ) that is used as a parameter to penalize violation of the side constraint budget. We find the best value for the Lagrange multiplier (μ) through a series of iterations of a new inner sub problem. We update the value of the Lagrange multiplier (μ) after each iteration.

DAD CSP Operator Model with Lagrangian Relaxation:

Additional Sets

$p \in P$ Iterations of Lagrangian relaxation (Note: $|P| = 25$ for our test networks)

Additional Parameters

μ_p Value of Lagrange multiplier in iteration p .

Additional Decision Variables

Z_p Objective function value during Lagrangian relaxation iteration l .

Formulation

$$Z_p = \min_{Y_{ijd}} \sum_{(i,j) \in A} \sum_{d \in D} (c_{ijd} + q_{ijd} \cdot \hat{X}_{ij}) \cdot Y_{ijd} + \mu_p \cdot \left[\left(\sum_{(i,j) \in A} \sum_{d \in D} t_{ij} \cdot Y_{ijd} \right) - time_budget \right], \quad (4.4a)$$

Subject to:

$$\sum_{j:(i,j) \in A} \sum_{d \in D} Y_{ijd} - \sum_{j:(j,i) \in A} \sum_{d \in D} Y_{jid} = \begin{cases} -1 & \text{if } i = s \\ 0 & \forall i \in N \setminus s, t \\ +1 & \text{if } i = t \end{cases}, \quad (4.4b)$$

$$Y_{ijd} \leq \hat{W}_{ijd} \quad \forall (i,j) \in A, d \in D, \quad (4.4c)$$

$$Y_{ijd} \geq 0 \quad \forall (i,j) \in A, d \in D, \quad (4.4d)$$

$$\sum_{\substack{(i,j) \in A: d \in D \\ \hat{Y}_{ijk}=0}} Y_{ijd} + \sum_{\substack{(i,j) \in A: d \in D \\ \hat{Y}_{ijk}=1}} (1 - Y_{ijd}) \geq 1. \quad (4.4e)$$

Equation (4.4a) shows the time budget constraint moved to the objective function with Lagrange multiplier (μ). Equations (4.4b) and (4.4c) are the same as (4.3b) and (4.3c). Equation (4.4d) is very different from its counterpart Equation (4.3f) because the time budget constraint has been moved to the objective function. The network structure of the problem has been regained, and the path variables (Y) may be treated as continuous variables. The network structure only allows integer solutions to occur, even though continuous variables are permitted. Equation (4.4e) restates Equation (4.3e).

The classic Lagrangian relaxation iterative model selects values for the Lagrange multiplier (μ) in order to maximize the lower bound of the operator model, as discussed in the introduction section. The solution to the Lagrangian relaxation of the inner sub problem of Equations (4.4a-e) occurs over a series of iterations (P) until a stopping

criterion is reached. The Lagrange multiplier (μ) begins with a value of zero for the first iteration. The multiplier is updated to a new value for subsequent iterations. The update of the Lagrange multiplier (μ) is essentially an adaptation of “Newton’s method” for solving systems of equations in sub gradient optimization (Ahuja et al., 1993, p. 612).

Lagrangian Relaxation Iterative Model:

Additional Parameters:

Lag_UB	Upper bound achieved in iterative Lagrangian relaxation
∇_p	Subgradient of the remaining time budget on a path in iteration l .
λ_p	Scalar multiplier for Lagrangian bounding heuristic in iteration l .
θ_p	Step length for update amount to Lagrange multiplier in next iteration.

Formulation:

$$L(\hat{W}, \hat{X}, \mu) = Z_p. \quad (4.4f)$$

$$\max_{\mu \geq 0} L(\hat{W}, \hat{X}, \mu), \quad (4.4g)$$

$$\|\nabla_p\| = \left(\sum_{(i,j) \in A} \sum_{d \in D} t_{ij} \cdot \hat{Y}_{ijd} \right) - time_budget, \quad (4.4h)$$

$$\theta_p = \frac{\lambda_p \cdot (Lag_UB - Z_p)}{\|\nabla_p\|^2}, \quad (4.4i)$$

$$\mu_{p+1} = \left[\mu_p + \left(\theta_p \cdot \|\nabla_p\| \right) \right]^+. \quad (4.4j)$$

We have applied the derivation of Ahuja et al. (1993) for Equations (4.4f-j) to update the Lagrange multiplier. Equation (4.4f) identifies the Lagrangian multiplier function (L), as defined in the introduction section. The objective function (4.4g) maximizes the Lagrangian multiplier function (L) through an iterative search for the best value of the multiplier (μ). Each increase in the value of the iterated solution to the system of Equations (4.4) updates the lower bound on the Lagrangian relaxation of the problem. If the solution of the current iteration of system of Equations (4.4) also satisfies the original side constraint of Equation (4.2g), then the upper bound of the Lagrangian relaxation can be updated. Equation (4.4h) represents the difference between the amount of time used on the shortest path and the time budget in the side constraint. Ahuja et al.

(1993) refer to this difference as a *subgradient norm* to update the Lagrange multiplier. After the inner decomposition sub problem is solved, we take the final values of the shortest path variables (Y) as fixed values to compute the sub gradient norm. If the sub gradient norm is positive, then the side constraint is violated. Likewise, if the sub gradient norm is negative, then the side constraint holds. Equation (4i) represents a *step length* for how much the value of the Lagrangian multiplier should be updated in the next iteration of Lagrangian relaxation. Equation (4.4i) approximates the step size of the change of the Lagrange multiplier (μ) (Fisher, 1981). Equation (4.4i) is a heuristic approach to Newton's method because the actual optimal value of the Lagrangian relaxation is not known in advance (Ahuja et al., 1993, p. 613). The scalar multiplier (λ) in the numerator of Equation (4.4i) can be any value, and we choose to start with 0.8 as suggested by Ahuja et al. (1993). The value of the scalar multiplier is reduced in half if the lower bound is not improved over three iterations of relaxation. In the heuristic, the current upper bound represents the best found feasible solution that does not violate the time budget. Equation (4.4j) is the update mechanism for the Lagrange multiplier (μ). Equation (4.4j) takes the positive part of the previous Lagrange multiplier plus the product of the step length and sub gradient norm, so that the value of the Lagrange multiplier is never negative.

Since no standard stopping criterion exists for the heuristic of Equation (4.4i), we terminate the updating of the Lagrange multiplier after 25 iterations if the upper and lower bounds do not match (Ahuja et al. 1993, p. 613). The choice of 25 iterations is arbitrary, but we observe in every case that almost no change to the solution occurs past this point. The Lagrangian upper bound is updated whenever the calculation in Equation (4.4h) yields a negative number, and the shortest path cost is lower than the current upper bound. The Lagrangian lower bound is updated when the shortest path cost is larger than all other iterations, regardless of the value of Equation (4.4h).

D. IMPROVEMENTS TO THE INNER DECOMPOSITION ALGORITHM

This section finds algorithmic improvements to the inner decomposition of DAD CSP with Lagrangian relaxation. Inner decomposition is based on fixed values of the defense variables (\hat{W}).

1. Lagrange Variable Problem Transformation

We devise a novel approach to calculate the best value of the Lagrange multiplier (μ) during inner AD decomposition. Our innovative approach eliminates the iterative process to determine the value of the Lagrange multiplier (μ) and decreases the amount of time needed to obtain a lower bound on the problem solution. Traditionally, the Lagrange multiplier (μ) is found via the iterative process shown in the previous section.

We change the Lagrange multiplier (μ) from a parameter that is determined iteratively, to a decision variable to be maximized. Another way to think about this change is to imagine that the attacker of the network gets to pick both the worst-case attack to the network as well as the value of the Lagrange multiplier (μ) that will be used by the defender. The attacker can choose a value of the Lagrange multiplier (μ) that maximizes the effect of his worst-case attack on the operation of the network. We call this change to the AD CSP problem *the Lagrange Variable AD CSP problem*.

Because Lagrangian relaxation is incorporated into the problem formulation, the solution obtained can be infeasible for the original side constraint. Note that the time budget is the side constraint in our formulation. The infeasibility possibility means that the Lagrange variable problem should not be used by itself to solve a problem instance, since the solution it finds may be either infeasible or suboptimal. However, the objective value obtained via Lagrangian relaxation is a valid lower bound on the original problem solution due to weak duality theory (Fisher, 1981). Thus, the Lagrange Variable problem can be an incremental step in a larger algorithm to find an optimal solution. We develop other techniques later in the chapter to obtain convergence on a final optimal feasible solution to the original problem instance. We formulate both the primal and dual versions of the AD CSP Lagrange Variable problem. A dual exists for the inner minimization of the path choice variables (Y) because of the Lagrangian relaxation.

In our formulation of the AD CSP Lagrange variable problem, we include dual variables on the constraints; these are the dual variables for any version of the Lagrangian relaxation with a fixed defense (W) and fixed attack (X), since the resulting problem is simply a shortest path linear program. These variables facilitate the formulation of a dual-ILP version of the AD CSP Lagrange variable primal problem.

AD CSP Lagrange Variable Primal Problem:

Additional Decision Variables:

μ	Lagrange multiplier variable for violating the time budget
π_i	Dual variable associated with network node i .
α_{ijd}	Dual variable associated with network arc (i, j) with defense d .

Formulation:

$$\max_{X_{ij}, \mu} \min_{Y_{jd}} \sum_{(i,j) \in A} \sum_{d \in D} (c_{ij} + q_{ijd} \cdot X_{ij}) \cdot Y_{jd} + \mu \cdot \left[\left(\sum_{(i,j) \in A} \sum_{d \in D} t_{ij} \cdot Y_{jd} \right) - time_budget \right], \quad (4.5a)$$

Subject to:

$$\sum_{j:(i,j) \in A} \sum_{d \in D} Y_{jd} - \sum_{j:(j,i) \in A} \sum_{d \in D} Y_{jd} = \begin{cases} -1 & \text{if } i = s \\ 0 & \forall i \in N \setminus s, t \\ +1 & \text{if } i = t \end{cases}, \quad (4.5b)$$

$$Y_{ijd} \leq \hat{W}_{ijd} \quad \forall (i, j) \in A, d \in D, \quad (4.5c)$$

$$\sum_{(i,j) \in A} r_{ij} \cdot X_{ij} \leq attack_budget, \quad (4.5d)$$

$$Y_{ijd} \geq 0 \quad \forall (i, j) \in A, d \in D, \quad (4.5e)$$

$$X_{ij} \in \{0, 1\} \quad \forall (i, j) \in A \quad (4.5f)$$

$$\mu \geq 0. \quad (4.5g)$$

We reformulate the inner AD model in (4.5a) with Lagrangian relaxation of the time constraint to the objective function. The noticeable change in (4.5a) is that the Lagrange multiplier (μ) is now a decision variable instead of a data parameter. Equation (4.5b) are the usual balance of flow constraints, and we introduce the dual variable ($-\pi$) to account for the activity at each node in the network. We introduce dual variables with a negative value to keep with the dual variable convention used in other chapters. Equation

(4.5c) introduces the dual variable $(-\alpha)$ to account for activity on each arc of the network. Equation (4.5d) is the unchanged restriction on the number of attacks. Equation (4.5e) allows path variables (Y) to be continuous because of the special network structure of the problem created by relaxing the time constraint to the objective function. Equation (4.5g) restricts the Lagrangian multiplier decision variable to positive values.

Since the inner minimization of AD CSP Lagrange Variable primal problem involves a continuous variable (Y) , we are able to solve it in at least two different ways. We can solve AD CSP Lagrange variable problem with Benders (1962) decomposition. A more attractive option would be to take the dual of the inner minimization problem and reduce the CSP AD Lagrange Variable problem from bi-level maxi-min optimization to a single level maximization problem. The advantage of taking the dual of the inner minimization is that the single-level maximization problem frequently can be solved more quickly than using Benders (1962) decomposition. This is the case for almost every instance of our problem that we discuss.

AD CSP Lagrange Variable Dual ILP Problem Formulation:

$$\max_{X_{ij}, \mu, \pi_i, \alpha_{ijd}} \left[- \sum_{i \in N} \left\{ \begin{array}{l} -1 \text{ if } i = s \\ 0 \quad \forall i \in N \setminus s, t \\ +1 \text{ if } i = t \end{array} \right\} \cdot \pi_i - \sum_{(i,j) \in A} \sum_{d \in D} \alpha_{ijd} \cdot \hat{W}_{ijd} \right] - \mu \cdot \text{time_budget}, \quad (4.5h)$$

Subject to:

$$-\pi_i + \pi_j - \alpha_{ijd} \leq c_{ij} + q_{ijd} \cdot X_{ij} + \mu \cdot t_{ij} \quad \forall (i,j) \in A, d \in D, \quad (4.5i)$$

$$\sum_{(i,j) \in A} r_{ij} \cdot X_{ij} \leq \text{attack_budget}, \quad (4.5j)$$

$$\pi_i \text{ unrestricted} \quad \forall i \in N, \quad (4.5k)$$

$$\pi_s \equiv 0, \quad (4.5l)$$

$$-\alpha_{ijd} \geq 0 \quad \forall (i,j) \in A, d \in D, \quad (4.5m)$$

$$X_{ij} \in \{0,1\} \quad \forall (i,j) \in A, \quad (4.5n)$$

$$\mu \geq 0. \quad (4.5o)$$

Expression (4.5h) is the single level maximization of the dual integer linear program (ILP). The first summation in (4.5h) has only one nonzero term (π_i) . Equation (4.5i) is the dual equation for each arc in the network. Equation (4.5j) is unchanged from

the primal problem. Equation (4.5k) and (4.5l) define the restrictions on the node dual variables, and anchors the start node to a value of zero. Equation (4.5m) ensures the arc variables are non-negative. Equations (4.5n-o) are unchanged from the primal problem.

The AD CSP Lagrange variable dual ILP problem is a mixed integer program that can be solved in one invocation of optimization software. The useful result of the AD CSP Lagrange variable dual ILP is the values of the attack variables (X) and the objective function value. The optimal values of the dual node and arc variables (π and α) are not of particular concern. However, there is no guarantee that the dual variables will be associated with a feasible path that satisfies the side constraint time budget. The Lagrange variable relaxation allows the time budget to be violated, which would permit solutions that are infeasible for the original problem instance.

We use the dual-ILP formulation for the remainder of the chapter because it avoids the confusion resulting from presenting a nested decomposition algorithm. In practice, we observe the dual-ILP is faster than decomposition of the AD CSP Lagrange variable primal problem on our test network problem instances. However, if Benders (1962) decomposition would be significantly faster than solving the dual-ILP for other problem instances, then it could be easily incorporated into our algorithm without a change to the theoretical results.

2. Path Enumeration Closes Lagrangian Relaxation Optimality Gap

When we choose to perform iteration to find the best value for the Lagrange multiplier (μ) in the relaxed operator model, we must address the issue of convergence. Lagrangian relaxation is not guaranteed to converge on an optimal solution. For example, Lagrangian relaxation of the CSP for the six node test network does not converge on an optimal solution when the time budget is 14 units as discussed in Ahuja et al. (1993, pp. 608–614). We require a method to obtain the optimal constrained shortest path when the Lagrangian relaxation bounds cannot agree. We employ a recursive path enumeration algorithm to close the gap between the Lagrangian upper and lower bounds, as described in Carlyle et al. (2008). We enumerate all of the time budget feasible paths that exist with costs between the Lagrangian upper and lower bounds with a recursive depth first search

network algorithm. Carlyle et al. (2008) show how this technique can be applied to medium and large size networks. The depth first search path enumeration algorithm we implement is adapted from Python Software Foundation (2003). Pseudo-code for the path enumeration algorithm is in the Appendix.

3. Multi-cut Inputs to AD Inner Master Problem

Each iteration of inner decomposition of the AD CSP problem involves finding the single best feasible path for a fixed defense plan in the inner sub problem. That path is attacked in the inner decomposition AD master problem. Inner decomposition could be done more quickly if multiple feasible short paths were fed to the master problem instead of just one path at a time. When the Lagrangian relaxation of the inner AD sub problem is iterated and the Lagrangian bounds do not match, there is an opportunity to find multiple feasible paths to close the Lagrangian optimality gap. The previous section described how path enumeration is used to close the gap between the Lagrangian upper and lower bounds of the relaxation of the AD sub problem by finding the shortest feasible path that exists between the Lagrangian upper and lower bounds. But, path enumeration also generates all of the feasible paths that have a cost between the Lagrangian upper and lower bounds of the relaxed AD inner sub problem. Instead of only utilizing the shortest cost path in between the Lagrangian upper and lower bounds found in the inner sub problem, we can utilize any subset of the feasible paths generated by the path enumeration algorithm in an attempt to reduce the number of iterations in the inner AD sub problem. Each feasible path generated through the path enumeration algorithm becomes an input cut to the inner decomposition AD master problem. Our use of more than one path per iteration as input cuts to the AD master problem are called *multi-cuts*.

In order to implement multi-cuts into the path enumeration algorithm, we only need to make one change in the final step of the path enumeration algorithm:

Return all time budget feasible paths with their associated costs.

This change means the CSP with Lagrangian relaxation inner AD master problem receives many feasible paths as inputs during an iteration. All feasible paths can now be given as input cuts to the AD master problem.

However, the use of multi-cuts does have a shortcoming. In large networks, it may be possible that an enormous number of feasible paths exist with costs that are between the Lagrangian upper and lower bounds. If the upper and lower bound gap is significantly large, an enormous amount of paths could become input cuts. We do not recommend the use of all multi-cuts on large networks since the constraint set could become prohibitively large. Instead, an analyst could limit the size of the constraint set by selecting only small number of multi-cuts to pass to the master problem.

Another criticism of the use of multi-cuts as inputs to the inner AD master problem is that using all feasible paths at once could generate some superfluous cuts in the AD master problem that do not influence the choice of the optimal attack solution. A rebuttal to this criticism is that it is also possible that some relevant cuts that would have been found in later iterations of inner decomposition may be discovered in earlier inner iterations because of the path enumeration routine. Therefore, we believe the benefit of finding relevant cuts earlier than normal through path enumeration outweighs the possibility of including some useless path cuts in the inner AD master problem.

Consider the six node test network. In this example, we assume zero defense budget, so no arcs are defended. We assume an attack budget of one unit, so only one arc may be attacked. We assume a time budget of 14 units. Next, we perform decomposition to obtain the best attack and the resulting shortest cost path subject to the time budget.

Lagrangian relaxation with path enumeration on the six node test network will take three iterations to find the worst-case attack and shortest path cost. The details of each iteration of Lagrangian relaxation with path enumeration are shown in Table 30. Figure 48 shows the final results of each iteration implemented on the test network. In Figure 48, each iteration is shown, starting in the top left. The final attack is shown with a red “X” and the shortest path chosen is a blue line for each iteration. The lower bound is obtained by the cost of the path with the initial attack, and the upper bound is obtained by the cost of the path with the final attack.

Table 30. Lagrangian Relaxation and Path Enumeration on Test Network

Iteration	Initial Attack	Shortest Path	Final Attack	AD Upper Bound	AD Lower Bound
1	None	(1,3), (3,2), (2,4), (4,6)	(1,3)	38	13
2	(1,3)	(1,2), (2,4), (4,5), (5,6)	(2,4)	38	14
3	(2,4)	(1,3), (3,2), (2,5), (5,6)	(2,4)	15	15

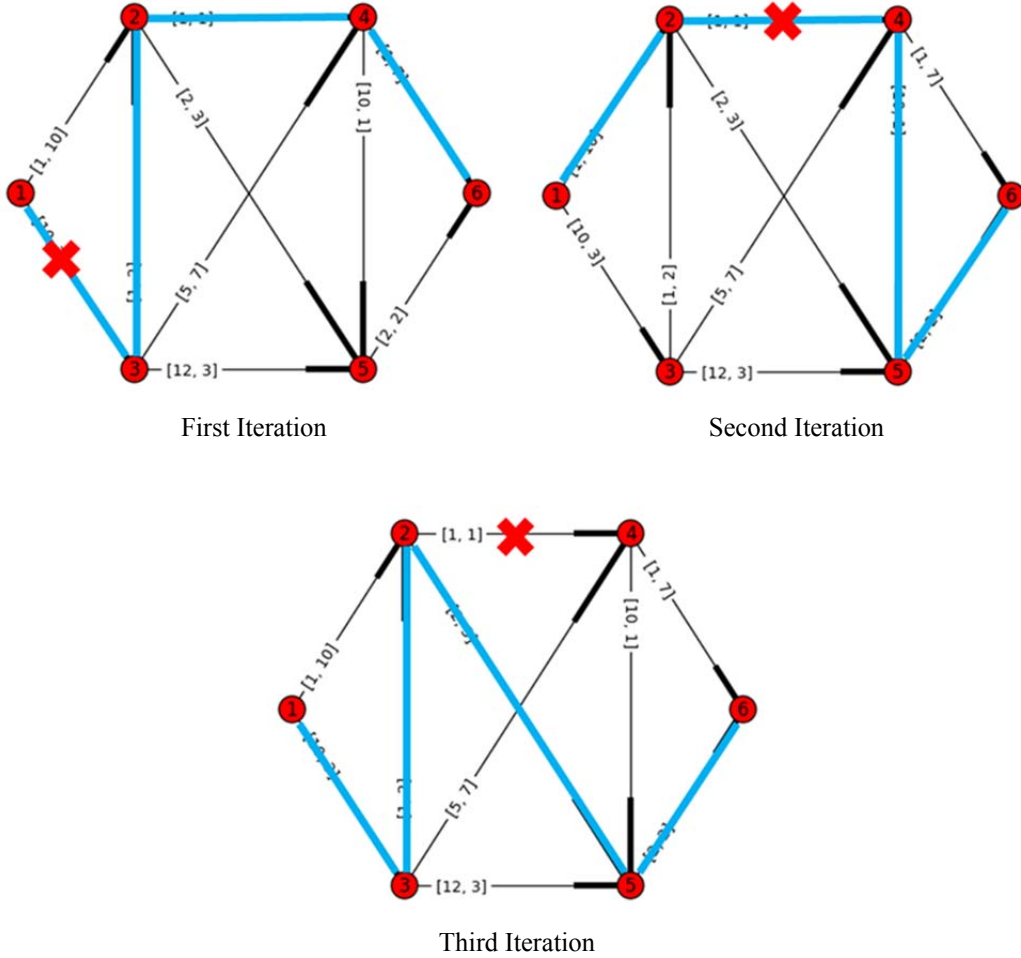


Figure 48. Lagrangian Relaxation and Path Enumeration on Test Network

Next, we examine the effect of introducing multi-cuts on the six node test network. The details of each iteration of Lagrangian relaxation with multi-cut path enumeration are shown in Table 31. Figure 49 shows the final results of each iteration implemented on the test network. In Figure 49, the first and second iterations are shown

from left to right. In the first iteration, the three feasible paths obtained from enumeration are shown with a solid blue line, a short dashed purple line, and a longer dashed orange line. The final attack based on all multi-cuts is shown with a red “X.” The second iteration only has one feasible path remaining.

Table 31. Multi-cut Algorithm Improvement on Six Node Test Network

Iteration	Initial Attack	Feasible Paths	Final Attack	AD Upper Bound	AD Lower Bound
1	None	(1,3), (3,2), (2,4), (4,6) (1,2), (2,4), (4,5), (5,6) (1,3), (3,2), (2,5), (5,6)	(2,4)	15	13
2	(2,4)	(1,3), (3,2), (2,5), (5,6)	(2,4)	15	15

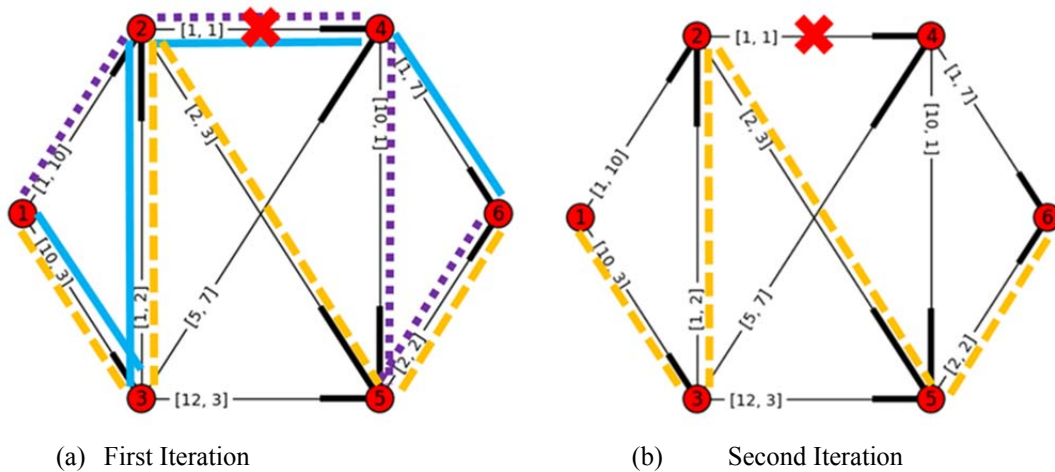


Figure 49. Multi-cut Algorithm Improvement on Six Node Test Network

The beneficial result of the inclusion of multi-cuts on the example problem instance is the number of iterations required to find the solution to the inner AD decomposition is reduced. The first iteration is able to enumerate three feasible paths before an attack is determined by the master problem. The worst-case attack can only affect two of the three feasible paths. When comparing the first iteration of the multi-cuts improvement with the three iterations of inner decomposition without the multi-cuts improvement, it can be seen that all feasible paths are obtained in one step with multi-cuts, and in this case, no superfluous paths were created. The multi-cuts improvement

eliminated the need to perform Lagrangian relaxation three times in order to obtain the three relevant paths for the AD master problem. There is only one feasible path found by enumeration in the second iteration because the other feasible paths from the first iteration are excluded. The penalty (q) imposed by the initial attack raises the cost of arc (2, 4) by an additional 25 units. The penalized cost of the arc results in paths that are more expensive than the upper bound.

E. RELAXATION IN INNER AND OUTER DECOMPOSITION

The Lagrange variable dual ILP problem of (4.5h-o) can be implemented into the AD inner decomposition in two different ways. The most powerful application of the Lagrange variable dual ILP problem is to completely replace AD inner decomposition with the Lagrange variable dual problem. Notice that both the Lagrange variable dual ILP problem and regular AD inner decomposition produce an attack (X) to provide as input to the outer decomposition. Regular inner decomposition can produce an optimal attack, but the Lagrange variable problem usually can produce an attack that is fairly close to optimal. There is no guarantee of the quality of the attack found by the Lagrange Variable dual ILP problem. Since optimal attacks are not found, this application is a heuristic approach. But, as we will show, the savings in computation time of not performing inner decomposition are quite dramatic.

A second application of the Lagrange variable dual ILP is as a *warm start* for the combination of the iterative Lagrangian relaxation operator model with path enumeration. In this second application, we use the fact that each round of inner decomposition begins with a starting point of no fixed attacks. Decomposition works by constructing attacks one at a time until the optimal attack is found. The Lagrange variable dual ILP problem can be used to find an initial attack that is usually fairly close to optimal. This initial attack from the Lagrange variable problem can then be used as a starting fixed attack (X) for Lagrangian relaxation of the operator model. The result is usually fewer inner decomposition iterations because the starting point attack is close to the optimal attack.

The path enumeration algorithm terminates with the constrained shortest path. The associated cost of the constrained shortest path is used to update the inner

decomposition lower bound. The values of the shortest path variables (Y) are fixed and supplied as input to the AD master problem.

The AD inner master problem used in conjunction with the operator model Lagrangian relaxation is unchanged from (3g–k). Each iteration of the AD master problem represents an opportunity to update the inner decomposition upper bound. When the inner decomposition lower bound is updated, the attack associated with that iteration is saved as the best known attack. We refer to this set of best attack variable values as *X star* (X^*). Also, the value of the Lagrange multiplier from the AD sub problem used in this iteration is saved as the best known Lagrange multiplier. We refer to this best known value of the Lagrange multiplier as *mu star* (μ^*). Pseudo-code for the implementation of the Lagrangian relaxation and path enumeration as the AD inner decomposition is presented in the appendix.

Either of the two approaches for inner decomposition can serve as a sub problem for the outer decomposition master problem. The outer decomposition master problem of DAD CSP with Lagrangian relaxation features the minimization of both binary defense variables (W) and path variables (Y). We use Lagrangian relaxation again on the outer decomposition master problem by moving the time budget constraint to the objective function. This transformation again restores the special network structure of the problem and allows us to treat the path variables (Y) as continuous variables. The Lagrange multiplier (μ) is necessary to regain the network structure of the outer master problem, and it can be different for each iteration of outer decomposition (k'). However, we do not treat the Lagrange multiplier (μ) as neither a decision variable nor an iterated value in the relaxed outer master problem. Instead, we choose to import the value of the Lagrange multiplier (μ) that was obtained from inner decomposition. We use the hat symbol (^) to denote the Lagrange multiplier is a fixed value in the relaxed DAD outer master problem.

DAD CSP with Lagrangian Relaxation Outer Master Problem:

Additional Parameter:

$\hat{\mu}_k$, Lagrange multiplier during iteration k' of outer decomposition.
 (Note: Obtained from inner decomposition)

Formulation:

$$\min_{W_{ijd}, Y_{jdk'}} Z_{DAD}, \quad (4.5p)$$

Subject to:

$$Z_{DAD} \geq \left[\sum_{(i,j) \in A} \sum_{d \in D} (c_{ijd} + q_{ijd} \cdot \hat{X}_{ijk'}) \cdot Y_{jdk'} \right] + \hat{\mu}_{k'} \cdot \left[\left(\sum_{(i,j) \in A} \sum_{d \in D} t_{ij} \cdot Y_{jdk'} \right) - T \right] \quad \forall k' \in K, \quad (4.5q)$$

$$\sum_{j:(i,j) \in A} \sum_{d \in D} Y_{jdk'} - \sum_{j:(j,i) \in A} \sum_{d \in D} Y_{jdk'} = \begin{cases} -1 & \text{if } i = s \\ 0 & \forall i \in N \setminus \{s, t\} \\ +1 & \text{if } i = t \end{cases} \quad \forall k' \in K, \quad (4.5r)$$

$$\sum_{(i,j) \in A} \sum_{\substack{d \in D: \\ d \neq d_0}} h_{ijd} \cdot W_{ijd} \leq \text{defense_budget}, \quad (4.5s)$$

$$Y_{jdk'} \leq W_{ijd} \quad \forall (i, j) \in A, d \in D, k' \in K, \quad (4.5t)$$

$$\sum_{d \in D} W_{ijd} = 1 \quad \forall (i, j) \in A, \quad (4.5u)$$

$$Y_{jdk'} \geq 0 \quad \forall (i, j) \in A, d \in D, k' \in K, \quad (4.5v)$$

$$W_{ijd} \in \{0, 1\} \quad \forall (i, j) \in A, d \in D. \quad (4.5w)$$

Expressions (4.5p) and (4.5q) work together to form the outer decomposition master problem to minimize the cost of shortest paths with fixed attacks by the choice of defense and path variables. Equation (4.5q) shows the Lagrangian relaxation of the time budget constraint is also present in the outer master formulation, which regains the network structure of the formulation. Equation (4.5r) is similar to (4.2b), except that iteration subscripts “ k ” have been added to the path variables for each outer decomposition. Equation (4.5s) restates (4.2f). Equation (4.5t) is similar to (4.2c), but the addition of subscript “ k ” allows for different path variables for each iteration of outer decomposition. Equation (4.5u) restates (4.2f). Equation (4.5v) allows path variables to have continuous values, since the Lagrangian relaxation regains the network structure. Equation (4.5w) is unchanged from (4.2j).

The new feature in the outer decomposition master problem is the Lagrangian relaxation of the time budget constraint into the iteration dependent objective equations (4.5q). Once again, the purpose of the relaxation is to reduce the number of binary variables that must be considered in the outer master problem. However, the effect of the

reduction is much more dramatic in the outer master problem because the path choice variables (Y) can be different in each iteration (k') of outer decomposition. Lagrangian relaxation has its greatest effect in the outer master problem because the number of path choice variables (Y) is dependent upon the number of iterations (k').

Determination of the values of the Lagrange multiplier ($\mu_{k'}$) in the outer decomposition is not trivial. The outer decomposition features the defense binary decision variables (W) in the constraints alongside the Lagrange multiplier (μ). Furthermore, the Lagrange multiplier (μ) can have a different value for each iteration of outer decomposition, since the path choice variables (Y) can be different in each outer iteration (k'). The outer master formulation means that the value of the Lagrange multiplier ($\mu_{k'}$) depends upon both the choice of defense variables (W) as well as relaxed path choice variables (Y), which also vary during each outer decomposition iteration (k').

We employ a novel feature to solve the issue of determining the Lagrange multiplier ($\mu_{k'}$) for each iteration of outer decomposition. We desire to avoid performing another round of Lagrangian relaxation iterations (L) to find the best value of the Lagrange multiplier for each round of DAD outer iterations (K'). We use the best value of the Lagrange multiplier from the inner decomposition (μ^*) as the only value of the Lagrange multiplier for the current iteration (k') in the outer decomposition. The rationale behind this choice is that the Lagrange multiplier (μ) serves as the amount of penalty that the objective function must incur for failing to adhere to the time budget constraint. The inner decomposition relaxation determined the best value for this penalty (μ) when defense variables (W) were fixed and the attack variables (X) were optimized.

The methodology for the updating of the outer decomposition upper and lower bounds is similar to the approach seen in Chapter I with the DAD minimum cost flow problem. Refer to Chapter I for the specifics of updating bounds.

F. TWO HEURISTIC ALGORITHMS FOR DAD CSP

In this section we utilize all of the mathematical models described in previous sections in order to create two related algorithms to quickly determine a reasonably good feasible defense (W), and establish a bound on the DAD CSP problem. The first

algorithm has the goal of obtaining valid upper and lower bounds on the DAD CSP problem instance and providing a good defense for the network that is not guaranteed to be optimal. We refer to the first algorithm as a *bounding* heuristic since its goal is to provide upper and lower bounds on a DAD CSP problem instance. The second algorithm has the goal of finding a valid lower bound and a good defense plan as fast as possible by ignoring the development of an upper bound. We refer to the second algorithm as a *speed* heuristic since its goal is to find a good defense plan and a lower bound as fast as possible. We use the term *good* defense plan because these heuristics have no guarantee of producing the optimal solution. We observe these heuristics could find the optimal solution for some instances, but they cannot guarantee optimality by themselves.

Using these heuristic approaches as a stepping stone to obtain the optimal solution will be explored in the next section. We have also included a second part of a name to both of these algorithms as a nod to their eventual use with another technique in the next section. We call both of these heuristic approaches *phase 1 algorithms* because by themselves they do not guarantee the ability to converge on an optimal solution of defenses (W) for a network. Rather, the phase 1 algorithms provide a starting point to help find the optimal solution more quickly than the regular nested decomposition approach discussed previously. Phase 1 algorithms also provide a known defense that is associated with the DAD lower bound. Both phase 1 algorithms can be called a heuristic approach to the DAD CSP problem because there is no guarantee of optimality, nor is there any guarantee on the solution quality that is obtained. However, these phase 1 algorithms both obtain at least one bound on a solution much faster than it would take to solve the DAD CSP problem via the regular nested decomposition approach.

1. DAD CSP Phase 1 Bounding Heuristic Algorithm

The phase 1 bounding heuristic algorithm incorporates all of the mathematical models shown in the previous sections. The Lagrange Variable dual ILP problem is used to produce an initial attack (X) that serves as a smart starting point for the inner decomposition AD sub problem. Multi-cuts are used from path enumeration to give multiple feasible attack cuts to the AD master problem. There are two nested loops in the

algorithm which represent the outer and inner decompositions of the DAD CSP problem instance.

Since our heuristic algorithm uses Lagrangian relaxation, there is no guarantee of convergence on an optimal solution. We introduce a stopping criterion for our heuristic algorithm to terminate whenever it repeats a defense (W) that it had found in a previous iteration. In our tests, we limit the number of outer iterations of the phase 1 algorithm at 199. We observe the algorithm always terminates because the defense plan is repeated twice before reaching 199 iterations.

DAD CSP Phase 1 Bounding Heuristic Algorithm:

Inputs:

- Network data with defense, attack, and time budgets
- Iteration limits for DAD (outer) and AD (inner) decomposition
- Optimality tolerances for all models (usually zero)

Outputs:

- DAD Upper & Lower Bounds, best known defense (W*) and attack (X2*).

Algorithm:

While DAD Upper bound – DAD Lower Bound \geq tolerance & DAD iteration \leq limit:

Solve AD CSP Lagrange Variable dual ILP

Obtain initial attack (X) and Lagrange multiplier (μ)

While AD Upper Bound – AD Lower Bound \geq tolerance & AD iteration \leq limit:

Solve AD CSP with Lagrangian relaxation inner sub problem

Obtain Lagrangian upper and lower bounds and best multiplier (μ)

If Lagrangian relaxation bounds match:

- Obtain shortest path (Y) & associated cost

Else:

- Perform *Recursive Path Enumeration Algorithm*

- Obtain all feasible paths (Y) & shortest path cost

If shortest path cost > AD (inner) Lower bound:

- Update AD (inner) lower bound

- Save associated attack (X) as best known attack (X*)

- Save associated Lagrange multiplier (μ) as best multiplier (μ^*)

Solve CSP Inner Decomposition AD Master Problem

Obtain attack plan (X) and associated cost

Update AD (inner) upper bound if associated cost decreases

Increment inner decomposition iteration counter

Update DAD (outer) Upper Bound with AD (inner) Upper Bound, if it is smaller

Solve DAD CSP Lagrangian Relaxation outer master

If outer objective function value > DAD master lower bound:

- Update DAD master lower bound

- Record values of defense variables as best defense (W*).

Save attack associated with this defense as (X2*)
 Increment DAD iteration counter
If current defense (W) matches a defense from a previous outer iteration:
 Exit outer while loop

In our testing of the phase 1 bounding algorithm, we limit the maximum value of inner decomposition iterations to one. We observe that limiting the inner decomposition to one iteration step has a positive benefit at the expense of a small degradation in precision. We can simplify the phase 1 bounding algorithm to obtain upper and lower bounds more quickly by limiting the AD inner decomposition to just one iteration. But, the bounds obtained from just one inner decomposition iteration may be less precise than if more iterations were performed. The one round of AD inner decomposition allows for all of the steps of the algorithm to be completed. The shortcoming of using only one iteration of inner decomposition is that the optimal attack (X) may not be found. However, we believe that the tradeoff of weaker bounds in this algorithm is justified by the increase in solution speed, since only one iteration of inner decomposition is performed for each iteration of outer decomposition. The phase 1 bound algorithm is a heuristic, so performing more than one iteration of inner decomposition may not have much benefit for the extra effort required in additional inner decomposition iterations. A secondary benefit of only having one iteration of inner decomposition is that we do not need to further complicate our algorithm by checking for repeated paths.

Proposition 4–2. The phase 1 bounding heuristic algorithm produces valid upper and lower bounds for a DAD CSP problem instance.

Proof of Proposition 4–2. The heuristic approach contains all of the constraint equations that appear in the original DAD CSP problem, except for the time constraint, which has been relaxed to the objective function. The time constraint only effects the choice of path variables (Y), as seen in Equation (4.1g). Each set of attack variable values (X) obtained in the heuristic are feasible attacks because they must obey the attack constraints of the original DAD CSP problem. Each set of defense variable values (W) obtained in the heuristic are feasible defenses because they must obey the defense constraints of the original DAD problem. Our heuristic approach is not concerned with

the values of the path variables (Y), so we need not worry if the values obtained for them via relaxation are feasible. Ahuja et al. prove via weak duality that for any value of the Lagrange multiplier (μ), the value of the Lagrangian function “is a lower bound on the optimal objective function value of the original optimization problem” (1993, p. 605). Our use of the same Lagrange multiplier value from the inner decomposition to the outer master problem produces the same effect of creating a valid lower bound on the overall DAD CSP problem. The phase 1 boundary algorithm also produces a valid upper bound. Since the phase 1 bound algorithm always includes at least one iteration of Lagrangian relaxation and path enumeration of the inner sub problem, we are guaranteed that the output of the inner sub problem is a feasible path (Y) for the original problem instance. The inner upper bound is computed by taking the best feasible path (Y) and the optimized attack (X) for a fixed defense (W). The inner upper bound is simply transferred to the outer decomposition as its upper bound if it is an improvement. Thus, the upper bound calculation is feasible solution to the original problem instance and therefore it is valid. Therefore, our use of Lagrangian relaxation in the phase 1 bound algorithm is guaranteed to produce valid upper and lower bounds on the original DAD CSP problem instance.

2. DAD CSP Phase 1 Speed Heuristic Algorithm

The phase 1 speed heuristic algorithm is a simplification of the phase 1 bounding heuristic algorithm. The speed improvement is centered on the question of the usefulness of the DAD CSP upper bound. Certainly, situations exist where feasible upper bounds are informative. But, when considering that the heuristic algorithm can be an incremental step toward obtaining the optimal solution within a larger algorithm, the importance of a valid upper bound is diminished. Since the optimal solution may be obtained after this incremental step, the importance of having a valid upper bound is diminished. A second, but less powerful concept against the importance of an upper bound is to consider the idea of a heuristic. The purpose of the heuristic is to obtain a good feasible solution to the original problem. The heuristic solution does not need to be optimal, but other qualities make a potentially suboptimal heuristic solution attractive. In this case, obtaining a feasible good suboptimal solution quickly to a DAD CSP problem is quite attractive.

We can modify the phase 1 bounding heuristic algorithm to be faster if we examine the inner decomposition to eliminate some of the calculations. The purpose of the inner decomposition is to determine a worst-case attack (X) as an input to the outer master problem. Consider that the phase 1 bounding heuristic algorithm obtains the worst-case attack (X) in general terms by performing three tasks. First, the phase 1 bounding algorithm uses the Lagrange variable problem to create a good guess initial attack to start the decomposition. The second task performs Lagrangian relaxation and path enumeration to find the shortest feasible path and an inner upper bound. The third task finds the worst-case attack and updates the inner lower bound. If we eliminate the second and third tasks from the inner decomposition, we can simplify the inner decomposition to just one task. Eliminating these two tasks is possible because the first task provides a reasonably good feasible attack (X) as inputs to the DAD outer master. The second and third tasks serve to ensure feasibility of a path based on that attack and calculate valid bounds. The third task may also find a better attack than the attack obtained by the Lagrange variable dual ILP, but it is not guaranteed. Elimination of the second and third tasks reduces the number of required calculations, which improves solution speed.

Elimination of the second and third major tasks from inner decomposition does have negative aspects. The only task remaining in inner decomposition is the Lagrange variable dual ILP problem, which can create an attack that is based on dual variables associated with an infeasible path. The eliminated task of Lagrangian relaxation with path enumeration ensures that a feasible path is found for the attack. Without a feasible path that is obtained with path enumeration, a feasible inner upper bound cannot be computed. The importance of the inner upper bound is that it is feasible solution to the original problem for a fixed defense (W). Therefore, the inner upper bound can be carried over as an upper bound on the overall DAD CSP problem instance.

DAD CSP Phase 1 Speed Heuristic Algorithm:

Inputs:

- CSP Network; defense, attack, and time budgets
- Iteration limits for DAD (outer) and AD (inner) decomposition
- Optimality tolerances for all models

Outputs:

DAD Lower Bound, best known defense (W^*), associated attack ($X2^*$).

Algorithm:

While DAD Upper bound – DAD Lower Bound \geq tolerance & iteration < max value:

Solve *AD CSP Lagrange Variable dual ILP*

 Obtain attack (X) and Lagrange multiplier (μ)

Solve *DAD CSP with Lagrangian relaxation outer master*

 Obtain and record values of defense variables (W)

 Obtain outer objective function value

If outer objective function value > DAD master lower bound:

 Update DAD master lower bound

 Record values of defense variables as best defense (W^*)

 Save attack associated with this defense as ($X2^*$)

If current defense (W) matches a defense from a previous outer iteration:

Exit while loop

G. EXTENDING THE HEURISTIC TO OBTAIN PROVABLY OPTIMAL SOLUTIONS

The phase 1 bounding heuristic algorithm in the previous section allows us to quickly obtain valid upper and lower bounds on the DAD CSP problem instance. The phase 1 speed heuristic quickly finds a lower bound, but no upper bound is provided. The shortcoming of both of these approaches is that phase 1 algorithms do not normally find the optimal solution to the DAD CSP problem. In this section, we use the results of the phase 1 heuristic as input to another algorithm to obtain the proven optimal solution to the DAD CSP problem. We call this algorithm the *phase 2 optimality algorithm* because it couples with one of the phase 1 algorithms from the previous section to produce an optimal solution to a DAD CSP problem instance. As the names imply, the combination of the phase 1 speed algorithm with the phase 2 optimality algorithm can find the optimal solution to a DAD CSP problem instance the quickest.

Simply stated, the phase 2 optimality algorithm is the regular nested decomposition discussed earlier in the chapter with two types of intelligent starting inputs which save time and iterations. First, regular nested decomposition begins the inner AD subproblem with no defense chosen as the initial fixed defense. We choose a smarter starting point that uses the best known defense (W^*) from the phase 1 defense as a starting point. We also include the attack associated with that best known defense as a

starting point for the inner AD subproblem. These smart inputs allow the first iteration of inner decomposition to benefit from the information that was learned by either phase 1 algorithm. Second, we observe regular nested decomposition starts with no information to build the outer master decomposition constraints. The phase 1 attack inputs to the phase 2 algorithm are called cuts because they act as constraints that trim portions of the outer master problem feasible region. Regular nested decomposition is completely dependent upon the results of inner decomposition to find a best attack to serve as a constraint for the outer master problem. The intelligent improvement is to use some or all of the best attacks found in each outer iteration of the phase 1 algorithms as input cuts to the phase 2 regular outer decomposition master problem.

The amount of phase 1 attack cuts supplied to the phase 2 algorithm is a choice for an analyst to make that is based on the characteristics of the network. Selecting all phase 1 attack cuts as inputs to the phase 2 outer master problem makes intuitive sense. All of the phase 1 attack cuts would create the most number of constraints that could trim the feasible region of the outer master problem. But, using all attack cuts as constraints for the outer master problem also has a significant drawback. The outer master problem is essentially a large binary linear program since both the defense variables (W) and path choice variables (Y) are binary. Each additional attack that is added as a constraint to the outer master problem will require another set of binary path variables (Y), because these variables have the outer iteration (K') as a subscript. A larger amount of binary variables can make the outer master problem more difficult to solve. Refer to (4.3m) and (4.3n) in the beginning of the chapter to see the iteration subscript on the path variables (Y) in the outer master problem. On the other hand, the analyst can choose to only pass a small number of phase 1 attack cuts to the phase 2 outer master problem. For example, the analyst could send only the two most recent cuts from the phase 1 problem. Sending a small number of attacks avoids the problem of a large binary linear program, but fewer input cuts results in less opportunity to trim portions of the outer master feasible region.

Through trial and error, we have two recommendations for the number of attack cuts to send from the phase 1 algorithm to the phase 2 outer master problem. When the network size is small or medium size, we include all of the phase 1 attacks as input cuts

to the phase 2 outer master problem. Our small and medium size test networks do not suffer tremendously from the problem of creating a very large amount of binary variables in the outer master problem when all of the phase 1 attacks are supplied as input cuts to phase 2. However, we recommend the opposite for very large networks. We find that the usage of the final two best attacks from the phase 1 algorithm are a good choice for input cuts to the phase 2 outer master problem for very large networks. This improvement gives the outer master some initial cuts to the feasible region of possible defense and path choices, while not creating too many binary variables overall. If all attack cuts are sent as inputs to the phase 2 outer master problem for very large networks, we observe that the speed of convergence is slowed down as the solver attempts branch and bound for a very large amount of binary variables. The real-world network example shows how the use of only two cuts from the phase 1 algorithm can be advantageous to the phase 2 algorithm.

DAD CSP Phase 2 OPTIMALITY Algorithm:

Inputs:

- Complete DAD CSP Phase 1 algorithm (either speed or bounding).
- Best defense (W^*) and associated attack ($X2^*$), number of iterations performed
- Choose a number of Phase 1 attack (X^*) to utilize as cuts
- Optimality tolerances for regular nested decomposition of DAD CSP

Outputs:

- Best defense plan (W^*), attack plan (X^*), and shortest path (Y)
- DAD outer upper & lower bounds

Algorithm:

- Use Phase 1 information to begin regular nested decomposition of DAD CSP:
 - Best defense (W^*) and associated attack ($X2^*$) is start point for first inner iteration AD inner decomposition sub problem
 - For** each best attack (X^*) **do**:
 - Create outer decomposition cut for the DAD outer master problem
 - Set DAD outer decomposition counter to Phase 1 iteration number

Proposition 4–3. Outer decomposition cuts from a Phase 1 heuristic are valid, feasible inputs to the regular nested decomposition of the Phase 2 optimality algorithm.

Proof of Proposition 4–3. Both Phase 1 heuristics create attacks (X), and a best known defense (W^*) that may be suboptimal. It is possible that a path (Y) obtained from the Lagrange variable problem may be infeasible, but the infeasibility is only limited to the path (Y) variables. The selection of attack variables (X) must obey the attack budget

constraint (4.2d), which is not relaxed. Therefore, the attack (X) from either Phase 1 heuristic algorithm is feasible in terms of the original problem instance. The best known defense (W^*) is always a feasible defense, even though it is obtained through a Lagrangian relaxation of the outer master in both Phase 1 algorithms. The best defense (W^*) is always feasible because it must obey the defense budget constraint of (4.2e), which is not relaxed. Therefore, both the attacks (X) and best defense (W^*) are feasible inputs to the regular nested decomposition of the phase 2 optimality algorithm.

Proposition 4–4. DAD CSP Phase 2 algorithm obtains the optimal solution to the DAD CSP problem.

Proof of Proposition 4–4. The Phase 2 optimality algorithm is nothing more than regular nested decomposition with the introduction of different starting points. Since regular nested decomposition does find optimal solutions, starting the algorithm from any other point in the feasible region will also lead to the optimal solution. Corner points of the feasible region are created as the cuts to the inner and outer master problems are added. The use of the phase 1 problem to start the phase 2 outer decomposition problem with more than zero feasible cuts to the feasible region does not affect the ability to reach the optimal corner point. Since each cut obtained in either phase 1 or phase 2 algorithms are feasible cuts, the optimal solution is never excluded from the feasible region. When the outer upper and lower bounds match, the optimal solution has been found.

H. RESULTS ON ARTIFICIAL TEST NETWORK

In this section we show the results on the 50 node grid network. We show the results of the phase 1 bounding algorithm to see how close its bounds can come to the optimal solution found in phase 2 optimality algorithm. We also show the results of the phase 1 speed algorithm combined with the phase 2 optimality algorithm in comparison to regular nested decomposition by itself in order to judge if our new algorithms improve the ability to obtain the optimal solution to a DAD problem instance more quickly. For the 50 node grid test network, we require all of our algorithms to converge on the optimal solution. We set all optimality tolerances for the 50 node grid test network to zero.

In order to understand how to interpret the result of a DAD CSP instance on the 50 node test network, we discuss the results for one budget scenario and the solution performance. Consider a DAD CSP problem instance on the 50 node grid network with defense and attack budgets of 5 units and a time budget of 40 units. The optimal defense, worst-case attack, and resulting shortest cost path as calculated by the regular nested decomposition approach are shown graphically in Figure 50 and summarized in Table 32. In Figure 50, the defended arcs are shown in blue dashed lines, the attacked arcs are shown with a red “X,” and the shortest path is shown as solid green lines. The shortest path cost is 27 units. The optimal solution is obtained via regular nested decomposition in 437.7 seconds. It is also possible that the optimal cost of 27 units could also be found through an equivalent optimal solution with different defenses, attacks, and shortest path.

Table 32. Optimal Defense, Attack and Shortest Path for DAD CSP Example

Defense Budget = 5 Attack Budget = 5 Time Budget = 40	50 Node Grid Test Network -- Arcs Selected Regular Nested Decomposition Method
Optimal Defenses	(1, 26) (26, 35) (28, 21) (33, 50) (35,28)
Worst-case Attacks	(15, 8) (23, 24) (26, 27) (32, 33) (40, 33)
Shortest Path	(1, 26) (15, 24) (22, 15) (24, 33) (26, 35) (28, 29) (29, 22) (33, 50) (35, 28)
Shortest Path Cost	27 units
Time to Solve	437.7 seconds

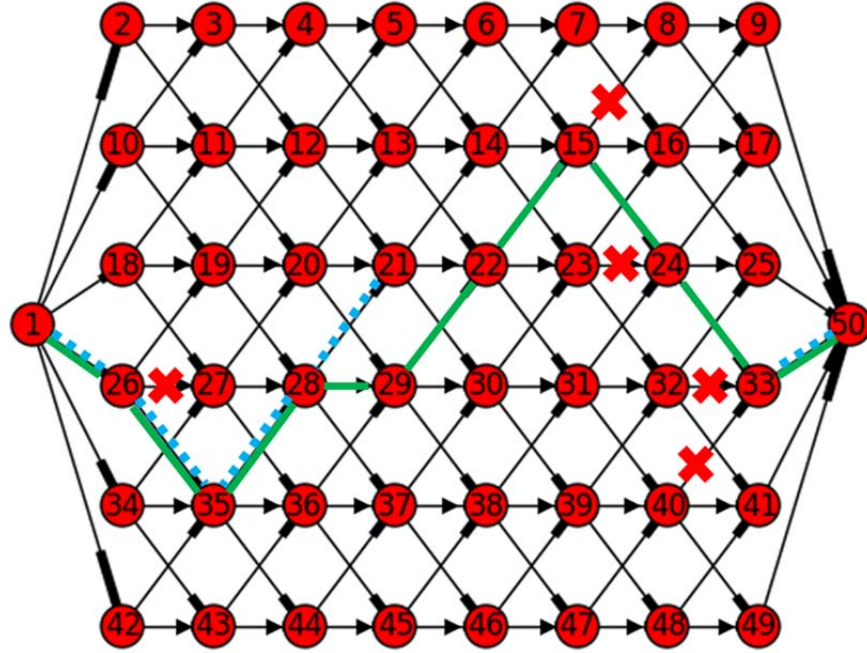


Figure 50. Optimal Defense, Attack and Shortest Path for DAD CSP Example

We observe the optimal defense protects the first few segments of the shortest path, along with the final segment. The worst-case attack appears to favor attacking many of the approaches to the final segment of the path. The worst-case attack also cuts off a low-cost arc at the beginning of the network, since arc (26, 27) has a cost of two units.

The results shown in Figure 50 can also be interpreted through the lens of game theory as the outcome of a sequential game. In this view, the defender acts first, the attacker acts second, and the defender acts third. First, the defender chooses to defend as much of the low cost path as possible, choosing the first four segments from the start node and the last segment connected to the terminal node. Second, the attacker uses most of his attacks to interdict many of the undefended approaches toward the last defended arc. The attacker chooses to penalize four of the paths to the terminal node, but he cannot attack them all. Since all of the paths cannot be attacked, he uses his last attack to cut off another low cost path near the beginning of the network by attacking arc (26, 27). Lastly, the defender (or operator) of the network must choose the shortest cost path that is available after the defenses and attacks have occurred. The defender (or operator) chooses the first three defended arcs as part of the shortest path, but the downstream

attacks in the network force him or her to abandon the fourth defended arc (28, 21). The downstream attacks make utilizing this defended arc no longer attractive in terms of cost. The defender then chooses undefended and unattacked arcs that provide the shortest path to get him or her to the final defended arc and the terminal node of the network.

We do not report results whenever the defense budget or attack budget is zero units. We observe that the phase 1 speed and phase 2 optimality algorithms work more slowly than regular nested decomposition whenever a defense or attack budget is zero units. Simplifying the problem to zero units in either the defense or attack budget changes the nature of the problem to something other than a DAD problem. An attack budget of zero units reduces the problem to a defender–defender (DD) problem, which is also known as the network design problem. Network design problems use other solution techniques that are well studied in the literature (Magnanti & Wong, 1984). A defense budget of zero units reduces the problem to an attacker – defender (AD) problem, which is the well-studied network interdiction problem, as previously discussed in the introduction. The network interdiction problem only requires one layer of decomposition to obtain a solution. Both of these zero budget cases are not our concern, since other existing procedures in the literature are tailored for these scenarios.

The next two subsections display the results obtained from both the phase 1 bounding algorithm and the phase 1 speed algorithm coupled with the phase 2 optimality algorithm for the 50 node grid network. The first subsection shows how quickly the phase 1 bounding algorithm can find usable bounds and defenses that are reasonably close to optimal. We also report when the phase 1 defense is also the optimal solution. The second subsection compares how quickly the optimal solution can be found for regular nested decomposition vs. the combination of the phase 1 speed and phase 2 optimality algorithms. We use two different methods to measure the effort it takes to obtain the optimal solution. Our results demonstrate that the combination of the phase 1 speed and phase 2 optimality algorithms obtain the same value for the optimal solution much faster than the regular nested decomposition approach.

1. Results for Combination of the Bounding and Optimality Algorithms

Table 33 compiles the results of the phase 1 bounding algorithm combined with the phase 2 optimality algorithm for defense and attack budget scenarios varying between one and seven units on the 50 node test network. We present the upper and lower bounds found with the Phase 1 bounding algorithm. We show the Phase 2 optimal solution value to verify that the optimal solution was found. We show the time it took the phase 2 optimality algorithm to obtain the solution based on the input it received from the Phase 1 bounding algorithm. We show the gap between the phase 1 lower bound and the optimal solution that was obtained in phase 2 to give an idea about the solution performance of the phase 1 bounding algorithm. We have shaded entries in this column green that have a gap within 10% of the optimal solution, yellow for gaps between 10%—25% and red for optimality gaps greater than 25%. The calculation of the optimality gap between the phase 1 lower bound and the optimal solution given in Equation (4.6a). Lastly, we have a column to indicate if the phase 1 bounding algorithm was able to find the optimal defense. We color code *yes* and *no* answers in the last column.

$$\text{Solution gap (\%)} = \frac{\text{Optimal Solution} - \text{Phase 1 Lower Bound}}{\text{Phase 1 Lower Bound}} \times 100\% \quad (4.6a)$$

Table 33. Upper and Lower Bounds from Phase 1 Bounding Heuristic

Defense Budget	Attack Budget	Phase 1 Solution (seconds)	Phase 2 Solution (seconds)	Phase 1 Upper Bound	Phase 1 Lower Bound	Phase 2 Optimal Solution	Phase 1 Bound vs. optimal gap	Phase 1 defense optimal?
1	1	9.3	5.7	47	19.49	22	12.9%	NO
1	2	10.8	10.5	76	21	25	19.0%	NO
1	3	12.4	14.7	60	24.93	30	20.3%	YES
1	4	23.7	37.7	60	21.4	34	58.9%	NO
1	5	32	20.3	60	32.92	35	6.3%	YES
1	6	18.1	24.7	97	42.16	43	2.0%	YES
1	7	9	37.7	195	45	45	0.0%	YES
2	1	17	5.3	45	19.49	20	2.6%	NO
2	2	11.9	13.4	72	21	23	9.5%	NO
2	3	13.5	19.2	60	24.93	27	8.3%	NO
2	4	29.4	87.2	41	21.4	32	49.5%	NO
2	5	41.3	81.3	40	30.4	33	8.6%	NO
2	6	21.4	51.9	86	31.94	36	12.7%	YES
2	7	19.1	100.5	54	29.05	39	34.3%	YES
3	1	19.1	5.4	45	19.49	20	2.6%	NO
3	2	25	23.8	72	21	23	9.5%	NO
3	3	32.4	12.2	53	24.21	26	7.4%	NO
3	4	61.9	50	35	26.32	29	10.2%	NO
3	5	55.7	102.5	35	28.16	32	13.6%	NO
3	6	35.1	52.4	85	31.1	34	9.3%	YES
3	7	89.4	231.1	54	33.01	37	12.1%	NO
4	1	16.4	3.6	45	19.16	20	4.4%	YES

Defense Budget	Attack Budget	Phase 1 Solution (seconds)	Phase 2 Solution (seconds)	Phase 1 Upper Bound	Phase 1 Lower Bound	Phase 2 Optimal Solution	Phase 1 Bound vs. optimal gap	Phase 1 defense optimal?
4	2	22.3	25.8	72	21	22	4.8%	NO
4	3	22.5	31	53	17	23	35.3%	NO
4	4	40.8	18	56	19.28	26	34.9%	YES
4	5	74.7	358.9	40	25.6	31	21.1%	YES
4	6	70.5	108.5	61	28.16	33	17.2%	YES
4	7	145.1	1451.7	51	33	34	3.0%	YES
5	1	21.4	4.1	45	19.16	20	4.4%	YES
5	2	27.7	18	72	20	22	10.0%	NO
5	3	40.5	16.7	53	21.29	22	3.3%	NO
5	4	100.2	61.4	41	23.67	25	5.6%	NO
5	5	138.4	271.9	45	25.6	27	5.5%	NO
5	6	141.7	354.7	60	28	29	3.6%	NO
5	7	197.7	1958.7	54	30.4	33	8.6%	NO
6	1	31.5	4.5	45	19.16	20	4.4%	YES
6	2	29.6	12.9	72	20	22	10.0%	NO
6	3	74.6	20	53	21.29	22	3.3%	NO
6	4	47.8	145	56	17.28	23	33.1%	NO
6	5	65.1	299.7	56	23.67	26	9.8%	YES
6	6	88.5	465.3	60	21.07	27	28.1%	NO
6	7	195.1	34283	54	27.5	31	12.7%	NO
7	1	33.2	4.5	45	17.87	20	11.9%	YES
7	2	21.5	15.7	47	20	21	5.0%	YES
7	3	43.7	10.7	53	20	22	10.0%	NO

Defense Budget	Attack Budget	Phase 1 Solution (seconds)	Phase 2 Solution (seconds)	Phase 1 Upper Bound	Phase 1 Lower Bound	Phase 2 Optimal Solution	Phase 1 Bound vs. optimal gap	Phase 1 defense optimal?
7	4	121	64.9	33	21.24	23	8.3%	NO
7	5	122.9	91.3	49	23	24	4.3%	NO
7	6	445.6	2360.6	45	25.12	26	3.5%	NO
7	7	354.3	11184	54	25.6	28	9.4%	NO

Table 33 shows that the phase 1 bounding algorithm is able to obtain valid upper and lower bounds on the DAD CSP problem instance quickly. It only takes the algorithm a few minutes to obtain bounds that are almost always within 25% of the optimal solution, and more often than not within 10% of the optimal solution. We observe that it can take considerably more time for phase 2 to prove the optimality of the solution to the original problem instance. The final column of the table shows that about half of the time, our phase 1 bounding algorithm finds the optimal defense. Recall that the phase 1 bounding algorithm is not guaranteed to find the optimal defense for a DAD CSP problem instance because it is a heuristic without a performance guarantee. The final column shows that it is possible that the optimal solution might be found by the phase 1 bounding algorithm. While optimal solution speed is not the goal of the phase 1 bounding and phase 2 optimal solution algorithm, this approach does have a solution speed that is comparable with regular nested decomposition.

2. Speed and Optimality Algorithms vs. Regular Decomposition

Table 34 provides the details of the speed comparison of the phase 1 speed algorithm coupled with the phase 2 optimality algorithm vs. the regular nested decomposition algorithm in system of Equations (4.3). The final column of Table 34 gives the relative performance percentage change between the phase 1 and 2 algorithms and regular nested decomposition. The final column entries are shaded green whenever the relative performance percentage is positive and shaded red when the relative performance percentage is negative. The calculation of the relative performance between the two algorithms is given in Equation (4.6b).

$$\text{Relative change (\%)} = \frac{\text{Nested Solution Time} - (\text{Phase 1+2 Solution Time})}{(\text{Phase 1+2 Solution Time})} \times 100\% \quad (4.6b)$$

Table 34. Solution Times for Phase 1 and 2 vs. Nested Decomposition

Defense Budget	Attack Budget	Phase 1 Solution (seconds)	Phase 2 Solution (seconds)	Phase 1+2 Solution (seconds)	Regular Solution (seconds)	Phase 1+2 Relative Change (%)
1	1	2.0	5.2	7.2	7.6	5%
1	2	3.2	5.8	8.9	12.9	44%
1	3	3.2	8.4	11.6	20.4	76%
1	4	3.7	11.3	15.0	52.7	251%
1	5	6.0	18.6	24.5	76.4	212%
1	6	4.4	26.9	31.3	105.7	238%
1	7	6.1	37.7	43.8	143.2	227%
2	1	3.0	3.0	6.0	11.2	87%
2	2	4.2	14.3	18.5	20.3	10%
2	3	4.4	26.2	30.6	28.8	-6%
2	4	12.0	14.6	26.5	121.9	360%
2	5	10.7	59.9	70.6	121.1	72%
2	6	6.2	51.3	57.5	170.4	197%
2	7	16.1	70.2	86.2	224.8	161%
3	1	4.0	3.3	7.3	11.9	63%
3	2	5.5	14.3	19.8	29.4	48%
3	3	8.1	7.2	15.3	34.4	124%
3	4	14.9	27.9	42.8	121.7	184%
3	5	13.4	95.6	109.0	251.3	131%
3	6	11.9	52.1	64.0	199.2	211%
3	7	40.1	348.0	388.1	467.0	20%
4	1	5.0	3.7	8.7	13.7	58%
4	2	10.1	25.3	35.4	35.2	-1%
4	3	6.7	6.3	13.0	54.2	316%
4	4	15.1	11.3	26.5	104.0	293%
4	5	28.0	435.0	462.9	390.9	-16%
4	6	33.1	79.2	112.3	626.3	458%
4	7	75.7	1151.1	1226.8	742.4	-39%
5	1	6.1	4.3	10.4	16.6	60%
5	2	10.0	17.7	27.7	38.5	39%
5	3	15.0	11.8	26.9	77.3	188%
5	4	15.0	64.6	79.6	147.3	85%
5	5	35.7	189.5	225.1	437.7	94%
5	6	63.5	189.1	252.6	496.9	97%
5	7	125.5	2522.2	2647.7	9845.6	272%
6	1	7.1	4.3	11.4	20.3	78%

Defense Budget	Attack Budget	Phase 1 Solution (seconds)	Phase 2 Solution (seconds)	Phase 1+2 Solution (seconds)	Regular Solution (seconds)	Phase 1+2 Relative Change (%)
6	2	12.4	9.9	22.3	57.5	158%
6	3	14.9	16.2	31.2	62.3	100%
6	4	31.3	20.9	52.2	140.3	169%
6	5	38.7	339.5	378.2	490.3	30%
6	6	139.8	337.8	477.6	990.1	107%
6	7	90.9	24836.9	24927.8	19843.5	-20%
7	1	8.2	4.5	12.7	21.2	67%
7	2	8.7	15.4	24.1	56.1	133%
7	3	14.3	12.8	27.0	75.6	180%
7	4	83.8	49.4	133.2	342.3	157%
7	5	67.9	207.5	275.4	836.1	204%
7	6	162.3	648.6	810.9	2257.0	178%
7	7	1138.0	1979.2	3117.2	10396.7	234%

Table 34 reveals that in almost all budget scenarios, the phase 1 speed and phase 2 optimality algorithms are faster than regular nested decomposition. There are only 5 out of 49 budget scenarios where the phase 1 and 2 algorithms perform slower than nested decomposition. In these few scenarios, the phase 1 and 2 algorithm is just slightly slower than regular nested decomposition, with a result of 39% slower in the worst case.

Recall that the objective of this chapter was to obtain an alternative solution methodology to regular nested decomposition of DAD CSP for large defense and attack budget scenarios that take a long time to obtain an optimal solution. The phase 1 speed and phase 2 optimality algorithms can be more than four times faster than regular nested decomposition! We frequently observe over 100% improvement in solution time.

The results of Table 34 have been plotted in Figure 51 for an easy visual comparison of the relative performance improvements of the phase 1 speed and phase 2 optimality algorithms in comparison to regular nested decomposition. The general contour of the three dimensional plot in Figure 51 appears to be rising as the defense and attack budgets increase. Figure 51 reinforces the notion that the phase 1 and 2 algorithms perform better than regular nested decomposition when the problem instance becomes more difficult to solve because of increasing defense and attack budgets.

Phase 1 & 2 vs. Nested Decomposition Solution Time Relative Performance

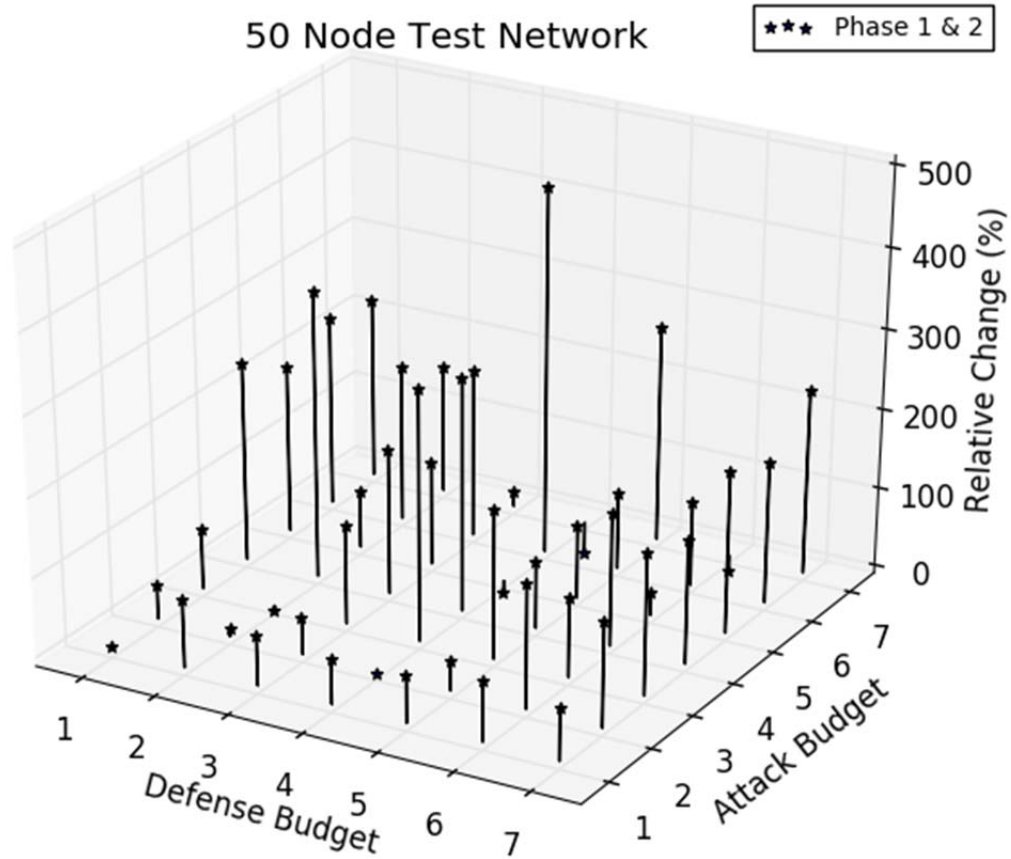


Figure 51. Relative Comparison of Solution Speed on 50 Node Test Network

A shortcoming of measuring time is the computing notion of wall clock time. Wall clock time is a combination of CPU time, input/output time and communication time in a computer. Wall clock time may be affected by other processes in the computer or by network issues. The use of wall clock time is the only way to measure the performance of nested decomposition in GAMS software. GAMS refers to each mixed integer program that it solves as a *model*. GAMS can measure the total wall clock time when it loads, builds, and solves a model. The ability to only measure CPU time in the GAMS software is limited to the actual solution time of a model (McCarl, 2015). Our DAD CSP algorithm implementation in GAMS code uses up to seven different models to solve a single CSP DAD problem instance in nested loops. In the phase 1 speed

algorithm, there are two models that work back and forth in a while loop. The AD dual ILP problem is a model and the DAD master is a model, both of which are called multiple times to establish a heuristic defense and lower bound. In the phase 2 optimality algorithm of the CSP DAD problem, there are five different models that work together to prove the optimal solution. The AD sub problem is a model, the AD sub problem with SEC is a model, the AD master is a model, the AD master with SEC is a model and the DAD master is a model. All of these five models in the phase 2 optimality algorithm are called multiple times in nested while loops in order to find the optimal solution to one DAD CSP problem instance. Thus, models are built many times and solvers are called over and over to for each problem instance.

An alternative way to measure the solution speed between the algorithms is to look at the number of iterations that it takes to find a solution. This view is advantageous because it is independent of the type of computer that is performing the computations. The phase 1 speed algorithm only has one iteration of the Lagrange variable dual ILP and the DAD outer master problem with Lagrangian relaxation for outer decomposition. The AD inner decomposition is eliminated by the use of the Lagrange variable dual ILP. This change is at the heart of why the phase 1 speed algorithm is able to provide an improvement in the amount of effort required to find a solution to a DAD CSP problem instance. The Phase 2 optimality algorithm, which is just regular nested decomposition has inner decomposition iterations for each outer decomposition iteration. Table 35 shows the difference in outer decomposition iterations between the phase 1 speed and phase 2 optimality algorithms vs. regular nested decomposition. Recall that regular decomposition refers to solving the DAD CSP problem instance with system of Equations (4.3). Problem instance scenarios with a defense or attack budget of zero units are not displayed. Equation (4.6c) is how the relative difference in the number of total iterations between the two approaches is computed.

$$\text{Iteration Relative Difference (\%)} = \frac{\text{Regular Iterations} - (\text{Phase 1+2 Iterations})}{(\text{Phase 1+2 Iterations})} \times 100\% \quad (4.6c)$$

Table 35. Iteration Comparison of Phase 1 and 2 vs. Nested Decomposition

Defense Budget	Attack Budget	Phase 1 “Speed” Iteration	Phase 2 Outer Iteration	Phase 2 Inner Iteration	Phase 1+2 Total Iterations	Regular Outer Iteration	Regular Inner Iteration	Regular Total Iterations	Iteration Relative Difference
1	1	2	2	9	13	2	10	12	-8%
1	2	3	1	10	14	2	18	20	43%
1	3	3	1	15	19	2	28	30	58%
1	4	3	1	21	25	3	71	74	196%
1	5	4	1	35	40	3	104	107	168%
1	6	3	1	51	55	3	139	142	158%
1	7	3	1	65	69	3	175	178	158%
2	1	3	1	4	8	3	14	17	113%
2	2	4	3	25	32	3	27	30	-6%
2	3	4	4	50	58	3	40	43	-26%
2	4	9	1	22	32	7	164	171	434%
2	5	6	4	118	128	5	164	169	32%
2	6	3	2	98	103	5	224	229	122%
2	7	4	2	109	115	5	270	275	139%
3	1	4	1	4	9	4	18	22	144%
3	2	5	3	23	31	5	44	49	58%
3	3	7	1	12	20	4	52	56	180%
3	4	11	2	37	50	9	180	189	278%
3	5	8	4	117	129	12	352	364	182%
3	6	5	2	89	96	7	297	304	217%
3	7	9	6	326	341	11	579	590	73%
4	1	5	1	4	10	5	21	26	160%

Defense Budget	Attack Budget	Phase 1 “Speed” Iteration	Phase 2 Outer Iteration	Phase 2 Inner Iteration	Phase 1+2 Total Iterations	Regular Outer Iteration	Regular Inner Iteration	Regular Total Iterations	Iteration Relative Difference
4	2	9	5	38	52	6	52	58	12%
4	3	6	1	9	16	7	78	85	431%
4	4	11	1	13	25	8	151	159	536%
4	5	15	4	106	125	15	403	418	234%
4	6	12	2	70	84	16	641	657	682%
4	7	18	7	357	382	17	918	935	145%
5	1	6	1	4	11	6	24	30	173%
5	2	9	4	25	38	7	55	62	63%
5	3	12	2	16	30	11	106	117	290%
5	4	11	4	58	73	11	192	203	178%
5	5	17	4	84	105	17	405	422	302%
5	6	21	3	90	114	16	552	568	398%
5	7	24	3	155	182	25	1118	1143	528%
6	1	7	1	4	12	7	27	34	183%
6	2	11	2	11	24	11	78	89	271%
6	3	12	3	20	35	8	88	96	174%
6	4	18	1	9	28	10	167	177	532%
6	5	18	6	102	126	15	302	317	152%
6	6	29	3	76	108	21	623	644	496%
6	7	20	19	737	776	30	1249	1279	65%
7	1	8	1	3	12	8	29	37	208%
7	2	8	4	17	29	12	68	80	176%
7	3	11	2	13	26	12	99	111	327%
7	4	26	1	9	36	18	233	251	597%

Defense Budget	Attack Budget	Phase 1 “Speed” Iteration	Phase 2 Outer Iteration	Phase 2 Inner Iteration	Phase 1+2 Total Iterations	Regular Outer Iteration	Regular Inner Iteration	Regular Total Iterations	Iteration Relative Difference
7	5	22	2	24	48	24	409	433	802%
7	6	29	2	43	74	26	678	704	851%
7	7	33	4	103	140	28	978	1006	619%

Table 35 shows how the use of phase 1 and 2 algorithms can outperform regular decomposition to solve DAD CSP problem instances. In all but three scenarios shown in Table 35, the Phase 1 and 2 combined algorithm has fewer decomposition iterations than the regular approach. In order to see the effect of the phase 1 speed algorithm, consider the comparison between phase 2 iterations and regular iterations. Recall that the phase 2 optimality algorithm is essentially regular nested decomposition, except that the outputs of the phase 1 speed algorithm are used as intelligent inputs to begin the phase 2 algorithm. One of the most impressive examples is the scenario of defense budget seven, attack budget six. In that scenario, 29 iterations of the phase 1 speed algorithm are performed to obtain a reasonable, but not provably optimal defense. The real improvement can be seen in comparison of phase 2 vs. the regular decomposition. The phase 2 algorithm only requires two outer iterations, but the regular approach requires 26 outer iterations. This effect is further amplified in inner decomposition, since phase 2 requires 43 inner iterations, but the regular approach requires 678 inner iterations. The absolute difference in the number of iterations between the phase 1 and 2 combined algorithm is 630 fewer iterations than the regular decomposition! Relatively speaking, the phase 1 and 2 combined algorithm requires eight times fewer iterations than the regular approach! Table 35 shows almost every other budget scenario has a similar, albeit less dramatic improvement in the reduction of the number of iterations needed to obtain the optimal solution to a problem instance scenario. The relative difference in iteration computation is quite remarkable, especially when one considers that a heuristic approach is responsible for the reduction in iterations.

The results of Table 35 have been plotted in Figure 52 for an easy visual comparison of the relative number of iterations performed between the combinations of the phase 1 speed and phase 2 optimality algorithms vs. regular nested decomposition. Figure 52 shows the phase 1 speed and phase 2 optimality algorithms require fewer iterations as the defense and attack budgets increase.

Phase 1 & 2 vs. Nested Decomposition Iterations Performed Comparison

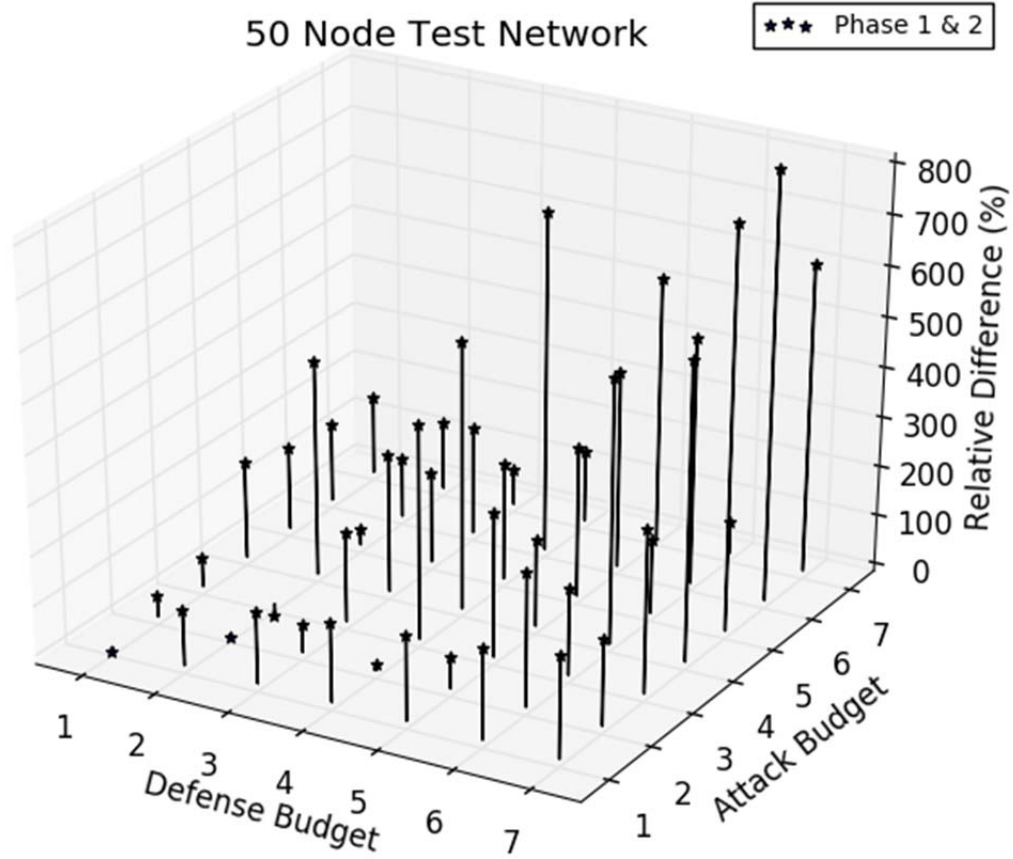


Figure 52. Relative Comparison of Iterations on 50 Node Test Network

I. APPLICATION OF DAD CSP TO A REAL WORLD NETWORK

1. Definition of Real World Network

We apply our DAD CSP solution procedures to a real world network. The real world network that we choose is the Maryland (MD), Virginia (VA) and Washington, DC (DC) transportation network that was used by Carlyle and Wood (2005). We obtained the data from these authors via personal communication. The network has 3,670 nodes and 9,876 directional arcs. This network includes interstate highways, federal highways and state routes, all of which we call *roads*. City streets and county thoroughfares are not included in the network. Nodes in this network represent various points along the roads. Each road is represented as two arcs between nodes, which allows for bi-directional

travel. We have two data items per arc. First, the distance in miles is given for each arc. Second, each arc has a speed limit in miles per hour. We convert speed limit and distance into a travel time for each arc by dividing distance by the speed limit. In contrast to our previous test networks, this real world network has cycles because roads allow for bi-directional travel. Figure 53 depicts the MD-VA-DC road network, with orange dots representing the nodes and blue lines representing the arcs. We choose a start node at the Pentagon and a termination node in Appomattox, VA, which are depicted with red stars.

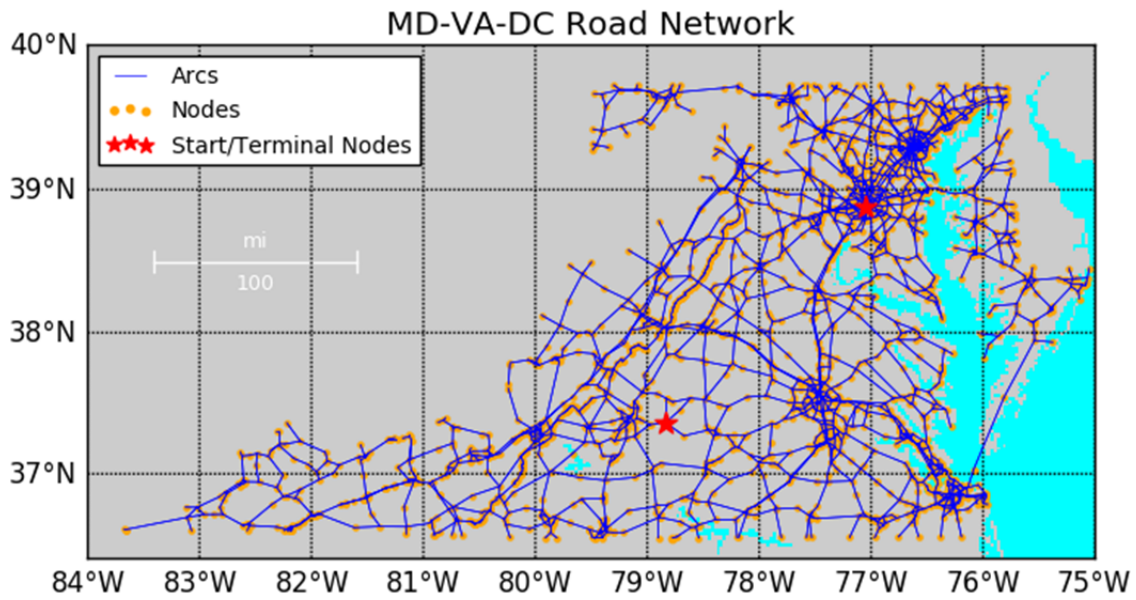


Figure 53. MD-VA-DC Road Network. Adapted from Carlyle and Wood (2005).

The start point of the problem instance is the highway adjacent to the Pentagon in Arlington, VA, and the termination point is a highway intersection in Appomattox, VA. Since many nodes and arcs are in the vicinity of Washington, DC, it is difficult to see the proximity of the starting node location. Figure 54 is a zoomed in view of the road network map for the Washington, DC, area. The start node can be seen in Figure 54 as a red star that is slightly southwest of the city center.

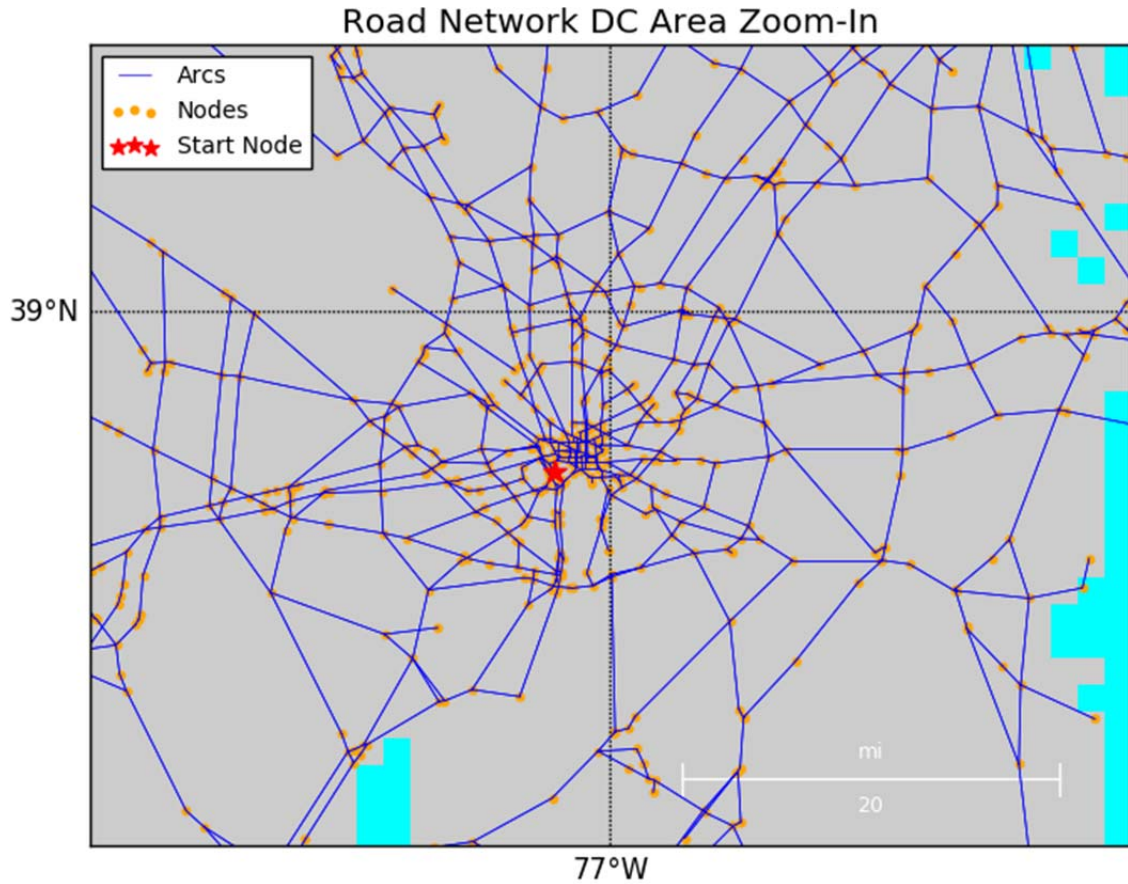


Figure 54. Road Network DC area. Adapted from Carlyle and Wood (2005).

The choice of start and termination nodes for our DAD CSP problem instance on a real-world network must be done with care. We define a *well-posed* DAD CSP problem instance to have a shortest distance path that is significantly different than the shortest time path. Similarly, we define an *ill-posed* DAD CSP problem instance to have a shortest distance path and a shortest time path that share most of the same arc segments. We observe that when attempting to constrain time when minimizing distance is the objective, it is possible to have an infeasible problem instance if the shortest time path is eliminated by enforcing a time budget. In the worst case, when the shortest distance path is also the quickest path, introducing a time budget of less than the quickest path may eliminate all potential paths from consideration. In less extreme situations of poorly chosen start and termination pairs, selecting a time budget to constrain the problem instance could result in a very small number of feasible paths, or paths with many

common arcs. Poorly chosen start and termination nodes could make the optimization of defenses and attacks in a DAD CSP problem instance uninteresting.

Our choice of route optimization from the Pentagon to Appomattox, VA is an example of a well-posed DAD CSP problem instance because the shortest distance path and the shortest time path do not have many common route segments. Our network data shows the shortest distance path is 170.58 miles long, which takes 3.65 hours. Figure 55 shows the shortest distance path with a solid pink line. The shortest time path is 3.332 hours, which takes 191.93 miles. The shortest time path is shown with a dashed brown line. The shortest distance and shortest time path route information is summarized in Table 36. We also include a Google Maps (2016) view of the area to orient our network maps with the real world in Figure 56. Note that the distances and times in our network data are different than those determined by Google Maps.

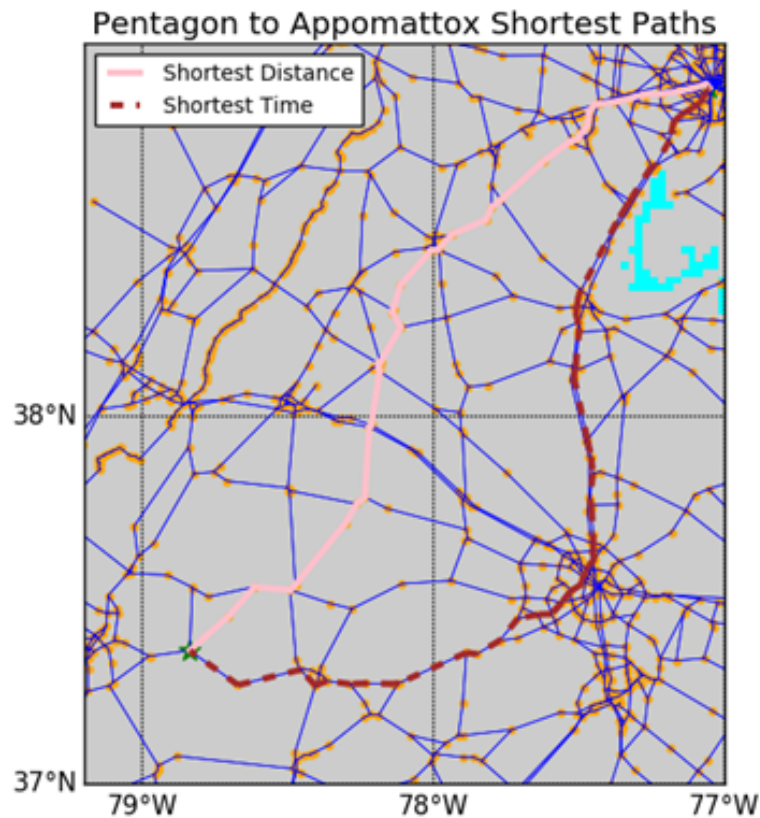


Figure 55. Shortest Distance and Time Paths from the Pentagon to Appomattox

Table 36. Shortest Distance and Time Paths from the Pentagon to Appomattox

Pentagon to Appomattox Path	Distance (miles)	Time (hours)	Description of Roads on Path
Shortest Distance	170.58	3.65	Washington Parkway to I-66 to US-15 to US-29 to VA-605
Shortest Time	191.93	3.332	I-395 to I-95 to VA-76/288 to US-360 to VA-307 to US-460

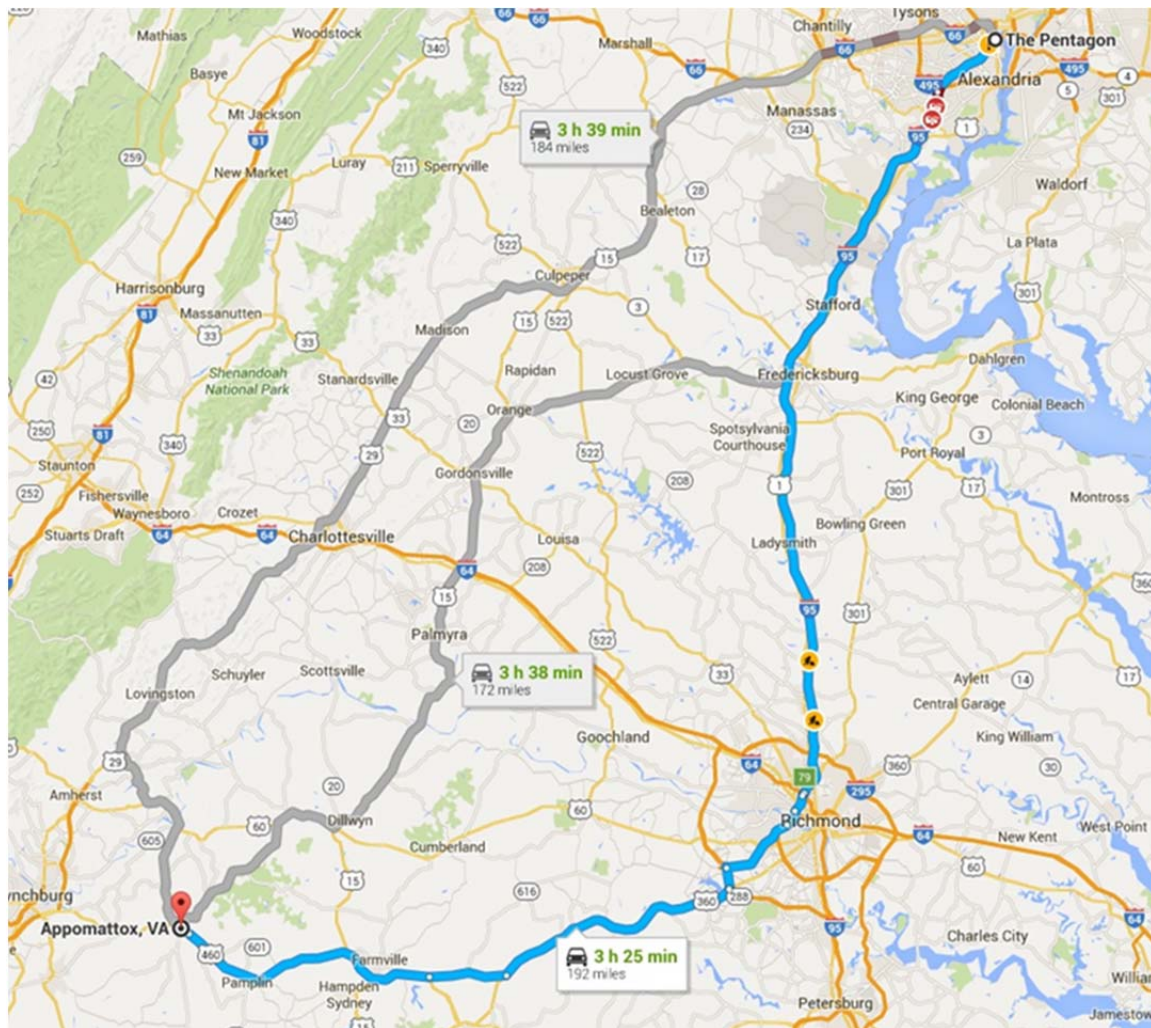


Figure 56. Map from the Pentagon to Appomattox. Source: Google Maps (2016).

2. DAD CSP Example Scenario on Real-World Network

An example DAD CSP problem scenario is to minimize the distance traveled from the Pentagon to Appomattox, VA, within a time budget of 3.6 hours, a defense budget of seven units and an attack budget of one unit. Notice that the time budget of 3.6 hours eliminates the unconstrained shortest path of 170.58 miles in 3.65 hours. A penalty (q) of 25 miles exists for traversing an attacked arc in the shortest path.

The results of the example scenario are shown in Figure 57. Our phase 1 speed heuristic gives an initial shortest distance path result of 180.99 miles. Recall the phase 1 result is not guaranteed to be optimal. The phase 1 algorithm gives a heuristic defense that also turns out to be the optimal defense in the phase 2 optimality algorithm. The white line segments represent defended arcs. The red line segment represents the one attacked arc, since the attack budget was only one unit. The phase 2 optimality algorithm is able to find the optimal solution for this particular instance, and it gives a result of 180.6 miles. The green line represents the shortest distance path. The blue line segments represent other arcs in the network, and the orange dots represent other network nodes.

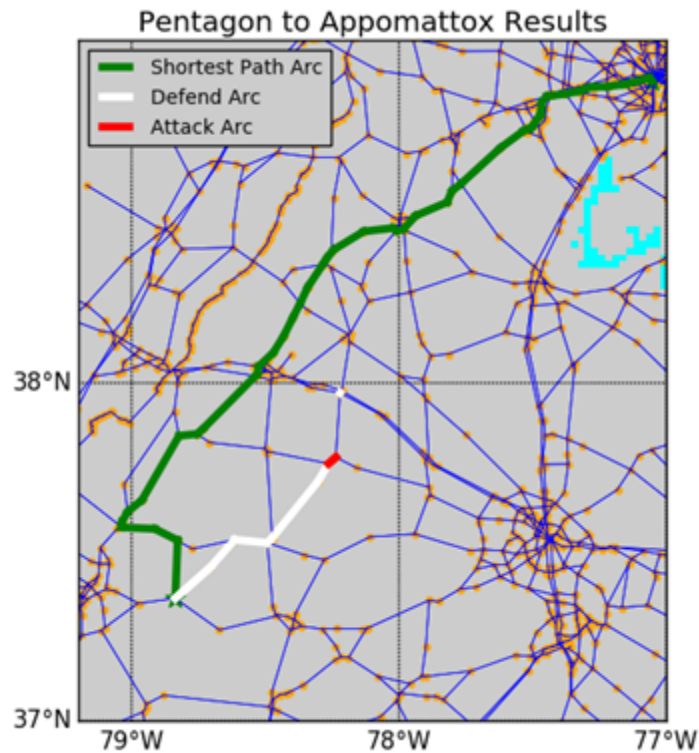


Figure 57. Example Problem for Optimal Defense and Attack of Road Network

Interpretation of the results of the example DAD problem instance can be inferred from Figure 57. The phase 1 speed algorithm identified a shortest distance path of 180.99 miles within the time budget of 3.60 hours. The defender chose to defend as much of the constrained shortest path as possible with the defense budget of seven units. By looking at the white line segments of defended arcs on Figure 57, one can visualize how that segment of the route would extend to form a 180.99 mile path. Next, the attacker desired to maximize the length of that constrained shortest path by attacking an arc segment that exists on that path. The attack action can be seen with the red line segment that is adjacent to the defended white line segments. Finally, the operator has to choose the constrained shortest path after the defenses and attacks occurred. Since a penalty (q) of 25 miles would be incurred for traveling on the shortest path found in the Phase 1 problem, the operator decides to choose the next shortest path in the network.

3. Algorithm Performance Comparison on Real-World Network

We test the scalability of both regular nested decomposition of DAD CSP as well as our phase 1 and phase 2 algorithms to large networks. We compare the performance capability with regular nested decomposition to gauge the effectiveness of our algorithm on a large network. We find that regular nested decomposition gets bogged down on the large size of the network when trying to resolve the final percentage points of an optimality gap. The phase 2 optimality algorithm struggles to converge on the optimal solution for problem scenarios when the attack budget is greater than one unit.

a. *Optimality Tolerances Required for Large Test Network*

The large size of the real world network forces us to accept the fact that a perfect optimal solution cannot be achieved in a reasonable amount of time when the defense and attack budgets are greater than two units. We observe that the majority of the time used by the IBM CPLEX solver is spent when the relative optimality gap between the best known solution and optimal solution is roughly 10% or less. In order to obtain a final solution to most problem instances in a reasonable amount of time, we accept that our solutions will not completely converge on the optimal objective function value. Termination prior to convergence on the optimal solution means that our approaches to solving DAD CSP for large-scale networks are approximation algorithms.

We set a relative optimality tolerance (RELTOL) for each of our mathematical models that are solved in nested decomposition according to some decision rules. Both the inner and outer decomposition loops have optimality tolerances that depend upon the solution quality of the models that are solved within them. We must choose all of the RELTOL values for each model with care. Failure to do so could result in a situation where a large RELTOL in one model makes it highly unlikely to ever satisfy a small RELTOL in its controlling decomposition loop. We set RELTOL values for each of the models of the DAD CSP real-world decomposition according to the following rules:

Relative optimality Tolerance (RELTOL) Equations:

$$\text{Overall DAD CSP RELTOL} = \text{Outer Decomposition RELTOL} \quad (4.7a)$$

$$\text{Inner Decomposition RELTOL} \leq \text{Outer Decomposition RELTOL} \quad (4.7b)$$

$$\text{Lagrange Variable dual ILP RELTOL} + \text{DAD Outer Master Lagrangian Relaxation RELTOL} \leq \text{Overall DAD CSP RELTOL} \quad (4.7c)$$

$$\text{AD subproblem RELTOL} + \text{AD Master RELTOL} \leq \text{AD Inner Decomposition RELTOL} \quad (4.7d)$$

$$\text{Inner Decomposition RELTOL} + \text{DAD Outer Master RELTOL} \leq \text{Outer Decomposition RELTOL} \quad (4.7e)$$

$$\text{AD Subproblem with SEC RELTOL} \leq \text{AD Inner Decomposition RELTOL} \quad (4.7f)$$

$$\text{AD Master with SEC RELTOL} \leq \text{DAD Outer Decomposition RELTOL} \quad (4.7g)$$

$$\text{Lagrange Variable dual ILP RELTOL} = \text{DAD Outer Master Lagrangian Relaxation RELTOL} \quad (4.7h)$$

$$\text{AD Inner Subproblem RELTOL} = \text{AD Inner Master RELTOL} \quad (4.7i)$$

$$\text{AD Inner Decomposition RELTOL} = \text{DAD Outer Master RELTOL} \quad (4.7j)$$

The overall RELTOL of a DAD CSP problem instance is limited by the value we set for the outer decomposition RELTOL, as shown in Equation (4.7a). Equation (4.7b) ensures that the inner decomposition RELTOL must be at least as strict as the outer decomposition. Otherwise, the outer decomposition may fail to converge. Equation (4.7c) describes the RELTOL requirements for the Phase 1 speed algorithm heuristic solution. We use the Lagrange variable dual ILP problem along the DAD master with Lagrangian Relaxation as a heuristic guess of the final problem solution, so the sum of their RELTOL should be at least as strict as our outer decomposition RELTOL. We observe that the tighter the value of RELTOL for the Lagrange Variable dual ILP problem may lead to better heuristic values that are closer to the optimal solution. Equation (4.7d) describes the RELTOL requirements of the inner decomposition. The inner decomposition consists of the AD inner sub problem and AD inner master problem. Both of these mathematical models must have RELTOL values that are smaller than the RELTOL for the inner decomposition, otherwise the inner decomposition itself may not converge. Equation (4.7e) describes the RELTOL requirements for the outer decomposition. The outer decomposition is comprised of the solution to the inner decomposition and the DAD

outer master problem. The outer decomposition RELTOL must be greater than the sum of the RELTOL from the inner decomposition and the DAD master problem, otherwise the outer decomposition may not converge. Equations (4.7f) and (4.7g) govern the RELTOL for activation of the SEC to prevent repeated paths or attacks, respectively. The bounds created by the problems with SEC are not valid for convergence of the original problem, so the RELTOL for these models can be less strict. We choose the RELTOL for the SEC models to be at least as strict as the decomposition RELTOL which activated the respective SEC in Equations (4.7f) and (4.7g), but this choice is arbitrary. Equation (4.7h) ensures both the sub problem and master problem of the Phase 1 speed heuristic both have the same RELTOL. Equations (4.7i) and (4.7j) ensure that the sub problem and master problem of both the inner and outer decompositions have the same RELTOL.

For our computation of DAD CSP problem instances on the real-world network, we set the RELTOL values as shown in Table 37. Since the largest RELTOL for any part of the DAD CSP problem is the outer decomposition RELTOL of 10%, the overall DAD CSP problem instance RELTOL is 10%. We note that relative tolerances are not strictly additive. Actual RELTOL limits are slightly larger than the values we choose. All test scenarios do not reveal any convergence issues with the RELTOL values shown in Table 37. From the example problem instance that was solved in the previous section, the optimal path took 180.6 miles and a 10% RELTOL equates to roughly 18 miles of travel.

Table 37. Relative Optimality Tolerances for DAD CSP

Algorithms	DAD CSP Problem Stage	RELTOL value
All	Overall DAD CSP Problem Instance	10 %
All	DAD Outer Decomposition	10 %
Phase 1 “speed”	Lagrange Variable Dual ILP Problem	5 %
Phase 1 “speed”	DAD Outer Master with Lagrangian Relaxation	5 %
Regular & Phase 2	DAD Outer Master Problem	5 %
Regular & Phase 2	AD Inner Decomposition	5 %
Regular & Phase 2	AD Master Problem	2.5 %
Regular & Phase 2	AD Subproblem	2.5 %
Regular & Phase 2	AD Master Problem with SEC	10 %
Regular & Phase 2	AD Subproblem with SEC	5 %

b. Performance Comparison

In order to make our combined algorithm as fast as possible, we must choose the number of phase 1 attacks to send to the phase 2 outer master problem. Since this real world network is very large, it is possible that sending too many attack cuts from phase 1 to phase 2 may bog down the outer master problem. We initially observe that using all available attacks from phase 1 as input cuts to phase 2 results in slower solution times for many budget scenarios. Instead, we choose to send only the two most recent phase 1 attacks to the phase 2 outer master problem. This choice allows us to take advantage of some of the feasible attacks in the phase 1 algorithm, and disregard the less helpful attacks that were generated in phase 1. We limit the number of attack cuts sent to phase 2 in order to help limit the amount of iteration dependent path variables (Y) in the outer master problem. The tables in this section reflect the use of two attack cuts shared between the phase 1 and phase 2 algorithms. Additionally, the best defense (W^*) and its associated attack are shared from phase 1 to start the decomposition in phase 2.

Table 38 shows the relative optimality gap comparison of the Phase 1 speed and phase 2 optimality algorithms compared to regular nested decomposition. We guarantee that the bounds achieved by our approach will be within 10% of optimality before we begin computation because of the RELTOL settings in the previous section. We shade the relative optimality gap green when the relative tolerance is less than 5% for a problem instance, and shade yellow when the optimality gap is between 5 to 10 %.

Table 38. Real World Network Solution Values and Relative Optimality Gap

Defense Budget	Attack Budget	Phase 1 Speed Bound	Phase 2 Upper Bound	Phase 2 Lower Bound	Phase 2 Optimality Gap (%)	Regular Upper Bound	Regular Lower Bound	Regular Optimality Gap (%)
1	1	182.38	184.64	182.38	1.24%	184.64	171.18	7.863%
1	2	187.37	196.80	187.37	5.03%	197.14	181.06	8.881%
1	3	196.18	199.71	196.18	1.80%	199.71	185.43	7.701%
1	4	205.99	212.64	206.38	3.03%	210.75	203.29	3.670%
1	5	212.49	220.69	212.49	3.86%	219.78	205.46	6.970%
2	1	182.38	184.64	182.38	1.24%	184.64	171.18	7.863%
2	2	187.37	196.80	187.37	5.03%	196.80	180.24	9.188%
2	3	194.49	199.71	194.49	2.68%	199.71	184.36	8.326%
2	4	204.66	210.60	204.66	2.90%	209.64	199.17	5.257%
2	5	209.37	217.07	209.37	3.68%	217.86	203.64	6.983%
3	1	182.38	184.64	182.38	1.24%	184.64	171.18	7.863%
3	2	187.49	196.80	187.49	4.97%	196.80	179.94	9.370%
3	3	194.51	199.71	194.51	2.67%	199.71	182.95	9.161%
3	4	200.50	209.64	200.50	4.56%	209.64	199.17	5.257%
3	5	208.26	217.50	208.26	4.44%	217.15	199.07	9.082%
4	1	180.99	184.64	180.99	2.02%	184.64	171.18	7.863%
4	2	186.01	197.14	186.01	5.98%	196.80	181.67	8.328%
4	3	193.86	199.71	193.86	3.02%	199.71	184.60	8.185%
4	4	200.50	209.64	200.50	4.56%	209.64	191.13	9.685%
4	5	209.33	217.88	209.33	4.08%	218.28	198.49	9.970%
5	1	180.99	184.64	180.99	2.02%	184.64	171.18	7.863%
5	2	186.01	197.14	186.01	5.98%	196.80	179.96	9.358%

5	3	193.86	199.71	193.86	3.02%	199.71	182.32	9.538%
5	4	199.07	209.64	199.07	5.31%	209.64	191.13	9.685%
5	5	208.07	218.58	208.07	5.05%	***	***	***
*** Indicates problem failed to converge after 86,400 seconds (24 hours)								

We observe an interesting effect from the phase 1 speed heuristic. The heuristic is able to find a valid lower bound that is always closer to optimal than the lower bound that is obtained from regular nested decomposition. The regular nested decomposition hardly ever generates an optimality gap less than 5% because it is not required to do so. Instead, the phase 1 heuristic generates only a lower bound, and sends it as an input to the phase 2 optimality algorithm, which has a 10% relative optimality tolerance requirement. Table 38 shows that the phase 1 and phase 2 approach always finds a bounds that are tighter than the regular nested decomposition approach. But, the phase 1 speed algorithm almost always generates a bound that can be proven to be within 5% of the optimal value, even though the requirement is a maximum 10% optimality gap. The phase 1 bound is almost always the lower bound in the phase 2 optimality algorithm.

Table 39 compares the time required to obtain a solution within the RELTOL parameters for the combination of the phase 1 speed and phase 2 optimality algorithms against regular nested decomposition. The computation of relative change percentage between the solution times for the two different solution approaches is the same as Equation (4.6b).

Table 39. Solution Time Performance Comparison on Real World Network

Defense Budget	Attack Budget	Phase 1 Solution (Seconds)	Phase 2 Solution (Seconds)	Phase 1+2 Solution (Seconds)	Regular Solution (Seconds)	Phase 1+2 Relative Change (%)
1	1	39.9	42.5	82.4	16.8	-80%
1	2	22.7	46.2	68.9	55.7	-19%
1	3	23.5	59.6	83.0	100.7	21%
1	4	42.1	87.6	129.7	468.3	261%
1	5	45.4	108.9	154.3	284.1	84%
2	1	67.8	33.4	101.1	17.9	-82%
2	2	83.8	51.7	135.5	187.8	39%
2	3	47.9	56.5	104.4	174.2	67%
2	4	125.4	95.3	220.8	1,516.3	587%
2	5	116.1	94.2	210.2	2,457.6	1069%
3	1	74.8	30.5	105.2	18.1	-83%
3	2	179.5	36.8	216.3	463.8	114%
3	3	131.1	67.2	198.3	246.4	24%
3	4	279.8	86.6	366.4	24,857.9	6684%
3	5	1,008.0	84.3	1,092.3	8,748.8	701%
4	1	108.2	32.1	140.4	18.2	-87%
4	2	135.0	47.6	182.6	1,019.0	458%
4	3	452.1	61.3	513.5	613.7	20%
4	4	1,735.7	98.2	1,833.8	15,771.7	760%
4	5	15,893.5	379.3	16,272.8	64,267.3	295%
5	1	172.2	46.2	218.3	15.5	-93%
5	2	300.7	48.0	348.6	502.9	44%
5	3	802.4	68.0	870.3	1,593.2	83%
5	4	14,890.1	104.3	14,994.4	30,020.9	100%
5	5	19,333.9	128.9	19,462.9	***	***
*** Indicates problem failed to converge after 86,400 seconds (24 hours)						

Table 39 reveals that the phase 1 speed and phase 2 optimality algorithm is quicker than regular nested decomposition in 19 of the 25 of the budget scenarios. Specifically, the phase 1 and 2 algorithms are always faster than regular nested decomposition whenever the attack budget is larger than two units. The amount of relative change between the two algorithms is dramatic. We observe whenever there is an improvement in solution time, our phase 1 and 2 algorithm is at least 20% faster, and is often much more than 100% faster than nested decomposition. The greatest example is

the scenario of defense budget of three units and attack budget four units, where our phase 1 and 2 algorithm is more than 60 times faster than regular nested decomposition.

Table 39 shows that regular nested decomposition is the faster approach whenever the attack budget is one unit. The slower performance with an attack budget of one unit may be attributed to two notions. First, the phase 1 algorithm requires at least two iterations in order to reach its stopping criterion of obtaining a repeated defense (W). So, the phase 1 algorithm must always spend the time to do at least two algorithms. In the case of 1 attack, this may be repetitive. Secondly, the branch and bound search for regular nested decomposition appears to be very simple, since it only takes about 15 to 18 seconds. Using notions about branch and bound of defenses that we developed in Chapter II, we believe that the outer decomposition need only defend against the best number of attacks that is equal to the defense budget. Further consideration of other attacks would not be necessary. Also, consider that regular nested decomposition starts the outer decomposition assuming no defenses as opposed to the phase 2 algorithm which starts with a heuristic defense from phase 1. In the simple case of only one attack, this usually smart starting defense may end up taking more time to pivot to a solution within tolerances than starting from the position of no defenses. We did not observe this behavior with the medium sized grid test network, but the test on the large network exposes this shortcoming of the phase 1 and 2 combined algorithm.

We notice a common theme for the performance of the phase 1 and 2 combined algorithm in Tables 38 and 39. For each budget scenario in Table 39, the bulk of the computing time is spent in the phase 1 speed heuristic in order to obtain a good estimate of a defense for the network. Then, a short amount of computing time is needed by the phase 2 optimality algorithm to prove that the defense found by phase 1 speed heuristic is within optimality tolerances. The amount of time to prove optimality in the phase 2 algorithm is usually one to two minutes, with one instance requiring roughly six minutes. We can tell that the defense and attack combination found by the phase 1 heuristic is usually optimal by looking at Table 38. The phase 1 bound is almost always the same as the lower bound found by phase 2 algorithm. The same value for the phase 1 bound and

the phase 2 lower bound means that the heuristic defense and attack found in phase 1 was proven to be the optimal defense and attack by the phase 2 algorithm.

We summarize the results of Table 39 graphically in Figure 58. We represent the relative change percentage for the scenario of defense and attack budgets of five units each as greater than 10,000%, since the regular nested decomposition problem failed to converge within acceptable tolerances after 24 hours of computation time. Figure 58 shows that our combined phase 1 and 2 approach is almost always significantly faster than regular nested decomposition of DAD CSP on the real-world network.

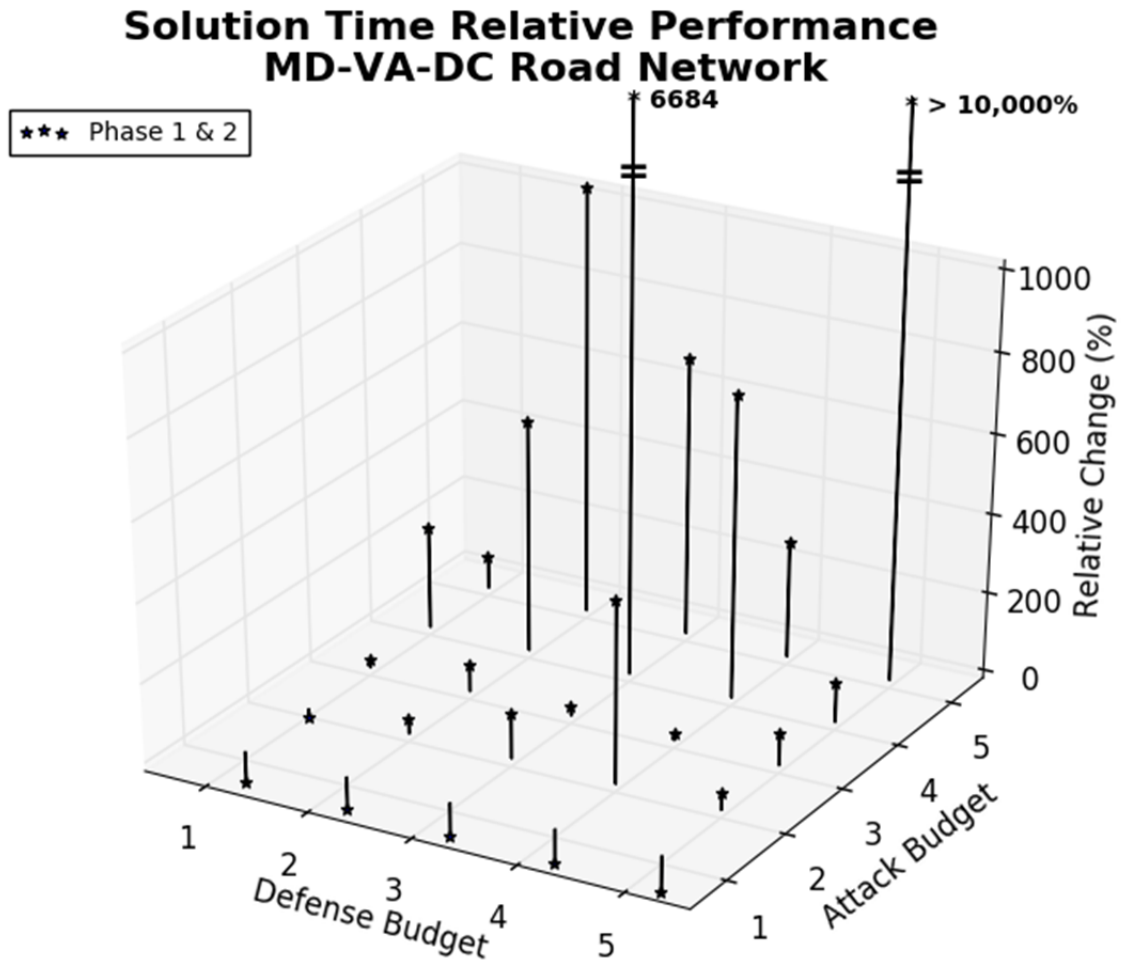


Figure 58. Solution Time Relative Performance for Road Network

There are two significant drawbacks in using solution time to measure the performance between the phase 1 and 2 algorithm vs. regular nested decomposition. The first shortcoming is the need to use the measurement of computer wall clock time as discussed in the previous section. Second, the implementation of our algorithm between the python programming language with interaction with the GAMS programming language may also result in some inefficiencies. The GAMS programming language has some inherent limitations such as the inability to call subroutines and dynamically create variable assignments. Because of these restrictions, we chose to implement the phase 1 and 2 algorithms within Python and use a call GAMS to perform all mathematical optimization. Each time our algorithm needs to solve an optimization problem, the python script of our algorithm must perform a variety of tasks in order to get an answer. For each call to solve an optimization problem, our script needs to write data files for use in GAMS, pause python execution, begin an instance of GAMS, read in data files created by Python, solve the model in GAMS, write data files for Python interpretation, exit the GAMS instance, resume the python script and read in the GAMS data files. This extensive use of file input/output by the computer was not noticeable when the models were the smaller test networks. But, in the large size real-world network, the sum of all of the small delays of switching between programs may be significant.

Because of the two shortcomings associated with measuring time in obtaining a solution, we next present an alternative measure of relative performance between the phase 1 and 2 algorithms vs. regular nested decomposition. We use the number of iterations performed as a proxy for the amount of effort required to obtain a solution within our specified tolerances for DAD CSP on the large-scale network. Table 12 presents the comparison of iterations performed between the phase 1 and 2 algorithm vs. regular nested decomposition. The calculation of the relative difference in the number of iterations performed is given by Equation (4.6c) in the previous section. We shade the relative iteration difference green when the combined phase 1 and 2 algorithm has a positive relative difference, and shade red otherwise.

Table 40. Comparison of DAD CSP Iterations for Real World Network

Defense Budget	Attack Budget	Phase 1 Speed Iteration	Phase 2 Outer Iteration	Phase 2 Inner Iteration	Phase 1+2 Total Iterations	Regular Outer Iteration	Regular Inner Iteration	Regular Total Iterations	Iteration Relative Difference
1	1	3	1	5	9	1	5	6	-33%
1	2	2	1	6	9	2	12	14	56%
1	3	2	1	11	14	2	24	26	86%
1	4	3	1	16	20	3	53	56	180%
1	5	3	1	23	27	3	59	62	130%
2	1	4	1	5	10	1	5	6	-40%
2	2	4	1	6	11	4	25	29	164%
2	3	3	1	11	15	3	37	40	167%
2	4	5	1	21	27	5	108	113	319%
2	5	5	1	19	25	5	122	127	408%
3	1	4	1	5	10	1	5	6	-40%
3	2	6	1	6	13	6	36	42	223%
3	3	5	1	11	17	3	37	40	135%
3	4	7	1	13	21	7	152	159	657%
3	5	11	1	19	31	7	157	164	429%
4	1	5	1	5	11	1	5	6	-45%
4	2	5	1	6	12	7	43	50	317%
4	3	8	1	7	16	4	50	54	238%
4	4	13	1	25	39	10	202	212	444%
4	5	25	1	25	51	10	172	182	257%
5	1	6	1	5	12	1	5	6	-50%

Defense Budget	Attack Budget	Phase 1 Speed Iteration	Phase 2 Outer Iteration	Phase 2 Inner Iteration	Phase 1+2 Total Iterations	Regular Outer Iteration	Regular Inner Iteration	Regular Total Iterations	Iteration Relative Difference
5	2	7	1	6	14	6	39	45	221%
5	3	9	1	11	21	5	59	64	205%
5	4	25	1	17	43	12	277	289	572%
5	5	25	1	24	50	***	***	***	***
*** Indicates problem failed to converge after 86,400 seconds (24 hours)									

The results of Table 40 are straightforward to interpret. Whenever the attack budget is greater than one unit, the phase 1 and 2 algorithm requires fewer iterations to solve DAD CSP than “regular” nested decomposition. The relative improvement in the number of iterations performed can be quite dramatic, with most values greater than a 100% improvement in the number of iterations performed. This measurement is an indication that the phase 1 and 2 algorithm requires less computational effort than regular nested decomposition whenever the attack budget scenario is greater than one unit. Conversely, we also observe that regular nested decomposition requires less computational effort only when the attack budget is one unit.

Figure 16 summarizes the results of Table 40. Again, we represent the relative difference percentage for the scenario of defense and attack budgets of five units each as greater than 1,000%, since the regular nested decomposition problem failed to converge within acceptable tolerances after 24 hours of computation time. Figure 16 shows that our Phase 1 and 2 approach requires significantly fewer iterations than regular nested decomposition of DAD CSP on the real world road network whenever the attack budget is greater than one unit.

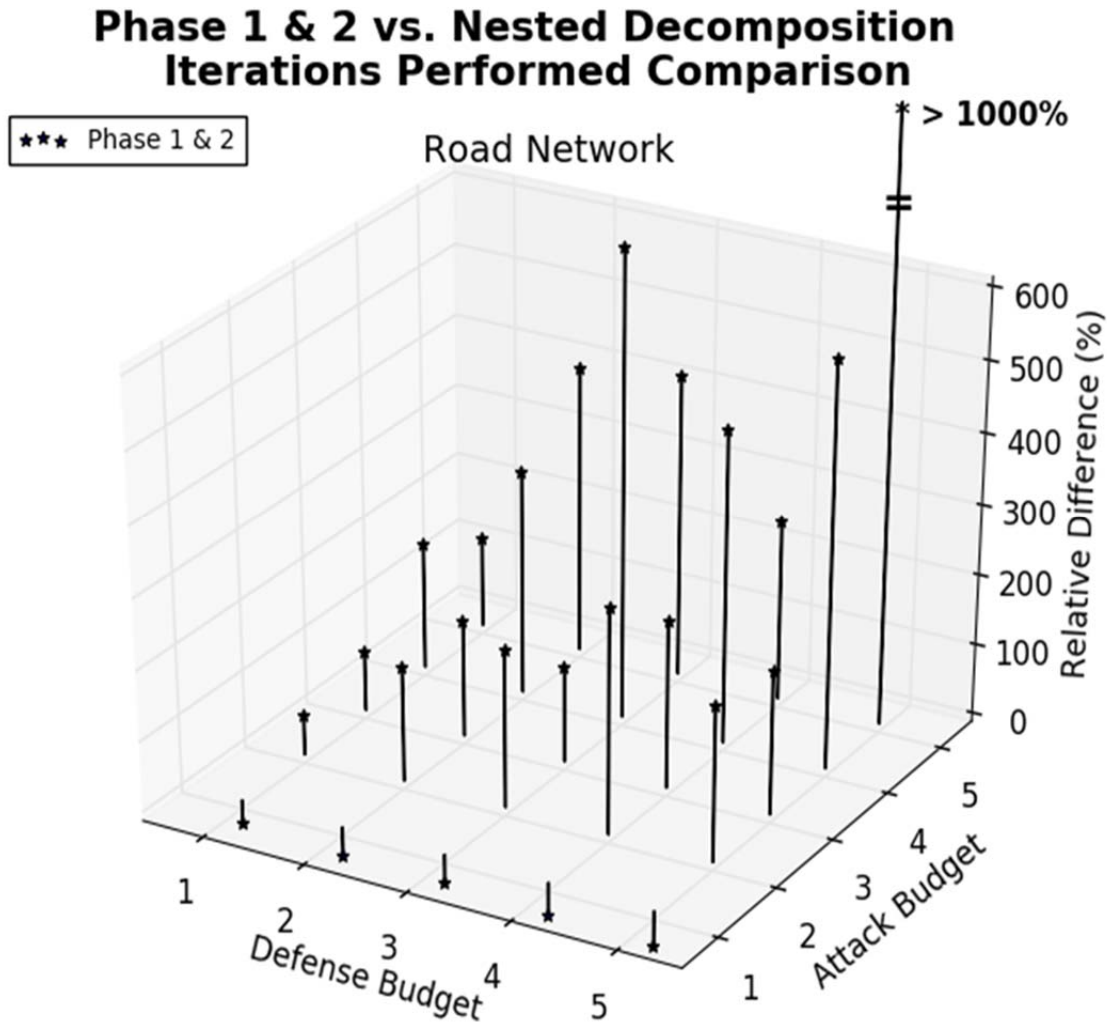


Figure 59. Relative Difference of Iterations Performed for Road Network

4. Lessons Learned from Real-World Network

We have shown that our approaches to DAD CSP are scalable up to large real world problems, albeit with some limitations. Comparison of DAD CSP solution times for the large real world network has given us more insight into the performance capability of both regular nested decomposition and our combined phase 1 and 2 algorithms. First, we learned that large networks take an inordinate amount of time to solve a DAD CSP problem instance to perfect optimality. In order to obtain DAD CSP problem instance results for a specific budget scenario in less than one day (86,400 seconds) we had to change DAD CSP to an approximation algorithm by introducing relative optimality

tolerances. We were able to obtain results in less than one day when the overall relative optimality tolerance was 10%. We discovered that the combined phase 1 and 2 algorithms have the potential to reduce the overall approximate solution time for a DAD CSP problem scenario, but an improvement in solution time is not guaranteed. We observe that our combined phase 1 and 2 algorithms perform the best when the problems are more difficult to solve because of large defense and attack budget scenarios.

J. SUMMARY

This chapter investigates the DAD CSP problem, which determines the optimal defenses, worst-case attacks and the resulting constrained shortest path between start and termination nodes in a network. The DAD CSP problem is NP-hard and has never been solved before in academic literature.

Initially, we showed how to apply regular nested decomposition procedures that are found in the literature for other DAD problems to solve DAD CSP. We observe that regular solution procedures become very slow on medium sized grid networks when the defense and attack budgets increase. We believe the slowdown of regular nested decomposition can be attributed to the combinatorial explosion of the number of feasible solutions as defense and attack budgets increase. The large number of feasible solutions experiences combinatorial growth because of the existence of binary defense, attack, and path choice variables for every arc in the network.

We applied the technique of Lagrangian relaxation to improve the speed of obtaining a solution to DAD CSP. We devised a novel *Lagrange variable problem* to perform bi-level Lagrangian relaxation in an innovative way. Instead of determining the best value of the Lagrange multiplier (μ) iteratively, we treat it as a decision variable to be maximized. The Lagrange variable problem can replace inner decomposition in order to form the core of what we call a *phase 1 speed heuristic* to find a reasonably good set of defenses and fairly close to optimal objective function value to a DAD CSP problem instance with no guarantee of solution quality. When the quickest time to obtain a solution to a DAD CSP problem instance is the primary concern, the phase 1 speed heuristic is the best choice.

Another technique to improve DAD CSP is Lagrangian relaxation combined with path enumeration. This approach uses the traditional iterative approach to find the best value of a Lagrange multiplier (μ). Since iterative attempts at Lagrangian relaxation may not converge, we investigated path enumeration to close the convergence gap. We found Lagrangian relaxation and path enumeration to be a slower approach at obtaining a heuristic solution. We proposed using *multi-cuts* to reduce the number of inner decomposition iterations. These techniques create what we call the *phase 1 bounding heuristic*. Even though it is a slower approach to obtain a heuristic solution, the bounding heuristic does improve the performance of solving DAD CSP problem instances.

It can take hours to find the optimal defense and objective function value for a DAD CSP problem instance when the defense and attack budgets are large. In a few minutes, our DAD CSP speed heuristic can produce a useful lower bounds on the optimal solution and a defense plan that is fairly close to the optimal solution. When the DAD CSP problem instance becomes difficult to solve due to a large defense and attack budgets, our combined phase 1 speed and phase 2 optimality algorithms have the potential to solve the problem optimally in a fraction of the time required by regular nested decomposition. We investigated how the choice of the number of attack cuts to pass from the phase 1 algorithm to the phase 2 algorithm can affect the speed of obtaining optimal or near optimal solutions.

Our use of a real-world transportation network proves that our solution procedures for DAD CSP can effectively scale to large problem sizes. We observe the nested decomposition approach requires an inordinate amount of time to converge on the optimal solution for large real world networks. We loosened the relative optimality tolerance to 10% in order to have reasonable convergence time. We learned that our requirements for an exact algorithm had to be changed to an approximation algorithm in order to accommodate the sheer size of the large network. We also reduced the number of input attacks from the phase 1 algorithm to the phase 2 algorithm in order to speed up the time to obtain a provably near optimal solution as much as possible. We are able to approximately solve DAD CSP problem instances on the large real-world network for scenarios of up to defense and attack budgets of five units using our combined phase 1

speed and phase 2 optimality algorithms within a relative optimality tolerance. The regular nested decomposition approach performed much slower than our combined phase 1 and 2 algorithm whenever the attack budget was greater than two units. We observe that our combined phase1 and 2 algorithm can perform much faster than regular nested decomposition on our real-world test network whenever the attack budget is greater than one unit. In the best case, our combined phase1 and 2 algorithm is more than 60 times faster than regular nested decomposition. But, we also discovered that phase 1 and 2 algorithms took more time than nested decomposition when the attack budget is only one unit. We observe that the majority of computational time and effort in our combined algorithm is required by the phase 1 speed heuristic and only a small amount of time is needed by the phase 2 optimality algorithm to confirm that the defense found is optimal or near optimal.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND FUTURE WORK

A. SUMMARY OF FINDINGS

Our dissertation examines the defender-attacker-defender (DAD) tri-level optimization model. We supply historical context to the problem of analysis of defenses and attacks to systems in an optimization model. We explain how existing literature solves a tri-level DAD model through the implementation of nested decomposition procedures for the DAD minimum cost flow model. We seek to expand knowledge of efficient solution procedures for the tri-level DAD optimization model through exploring many interesting areas of the problem and creating new algorithms to improve the ability to solve problem instances. Our research investigated implicit enumeration (IE) and nested defenses for the DAD minimum cost flow problem, and also presented heuristic and optimal solution procedures for the DAD constrained shortest path (CSP) problem.

We show how to apply known IE procedures to the DAD minimum cost flow problem. We expand upon current literature for the building of an enumeration tree for the DAD model by determining how to include additions to networks as potential defenses. IE is an efficient way to find either all multiple equivalent optimal solutions or all solutions within a fixed percentage of the optimal solution. IE allows us to transform a complicated tri-level DAD model into a sequence of abridged bi-level AD models.

Our IE algorithm for the DAD problem instance initially restricts the defense budget to zero units and solves a bi-level AD model to observe worst-case attacks. From the root node, an enumeration tree is generated by systematically relaxing this initial restriction. The relaxation occurs as an arrangement of defenses against to the worst-case attack by an incremental increase of the defense budget. Branches of the tree are formed by each defense against the worst-case attack that improves network performance. Defenses can either be hardening existing edges to prevent an attack, or adding a new edge to the network that would improve flow. Nodes at the end of branches are new instances of bi-level AD problems with fixed defenses that are represented by the branches of the tree from the root to that node. Our IE algorithm ignores defenses that

cannot improve the system based on this attack. Ignoring defenses that do not improve the system in light of the worst-case attack function is a pruning rule to keep the size of the enumeration tree as small as possible. We develop theory behind our pruning rule for adding new edges as defenses in detail. The process of solving fixed defense bi-level AD models, observing worst-case attacks and defending against them by incrementally increasing the defense budget repeats at each new tree node and continues until the defense budget is exhausted. An enumeration tree is created that contains all optimal and near optimal solutions to the original DAD minimum cost flow problem instance.

Our IE algorithm for the DAD problem can provide unique benefits for a decision maker. Equivalent optimal solutions are a useful resource for a decision maker that must consider factors that are beyond the scope of mathematical modeling when considering the best defense choices to make for an existing network. It is possible that a decision maker may not be indifferent between two or more mathematically equivalent solutions when other real-world considerations are taken into account. It is also possible that a near optimal solution that is found efficiently with IE could be attractive to a decision maker because of the tradeoff between optimality and other factors which cannot be modeled mathematically. Secondly, IE can also provide an alternative method to solve a DAD problem instance when a nested decomposition approach may be impractical or intractable. IE provides the ability to reduce the complexity of a tri-level optimization problem to a series of simpler bi-level optimization problems. In the case of a difficult tri-level DAD problem, IE offers the analyst an alternative solution algorithm.

Our research into nested defenses for the DAD minimum cost flow problem quantifies the additional *cost* incurred by requiring a prioritized list of defenses over a series of budget scenarios. Nested defenses are a monotonic sequence of sets where each set of defenses for a particular budget scenario contains the set of defenses for the smaller scenario. This type of circumstance can occur when defense budgets, attack budgets, or both are unknown when decisions about system defenses must be made. We show how to implement nesting constraints to the DAD minimum cost flow model. Our research shows how the collection of objective function values corresponding to a set of nested

defenses are almost always worse than the set of optimal objective function values for a range of DAD problem instances when budget scenarios are parameterized.

Our research into nested defenses on the DAD minimum cost flow problem gives both heuristic approaches and a parametric programming approach to find a set of nested defenses for a range of budget scenarios. Our heuristic approaches involve finding an optimal solution to a DAD problem instance for one budget scenario and using that result as a starting point to construct defenses for other budget scenarios. Heuristic starting points for nested defenses are investigated for minimum, midrange, and maximum budgets. We apply vector and matrix norms as well as other decision analysis tools to provide an indication of how to compare the performance of different nesting strategies against each other and against the set of optimal solutions. But, there is no guarantee that the heuristic approach will produce a set of nested defenses that are as close as possible to the set of optimal solutions for a range of budget scenarios. We show that it is possible to have a set of nested defenses that are suboptimal for every budget scenario, and that set can be closer to the set of optimal non-nested solutions than all of our other heuristic nesting strategies. We then reformulate the problem to find the closest possible set of nested defenses to the set of optimal non-nested solutions in terms of a vector norm. We consider a parametric programming approach to determine the best set of nested defenses to a DAD minimum cost flow instance. We show that parametric programming finds the closest possible nested defense to the set of optimal non-nested defenses based on a vector norm measurement for a sequence of uncertain defense and attack budget scenarios.

Our research into the DAD Constrained Shortest Path (CSP) problem shows how tri-level optimization modeling can expand from network flow models into binary programming. In this model, arc or edge additions are not allowed. The addition of the side constraint takes away the network structure of a traditional shortest path problem, and we explain how that change means the DAD CSP model is not a network flow problem. We show how to apply traditional nested decomposition to the DAD CSP problem and that it is possible to achieve results for some problem instances using this approach. However, we demonstrate that nested decomposition procedures gets bogged

down in the binary variable branch and bound procedure when the defense and attack budgets increase for medium-sized grid networks.

We introduce Lagrangian relaxation of the side constraint for both the operator and defender models of DAD CSP in order to regain the network structure of the problem. We investigate two different ways to incorporate the Lagrangian relaxation of the side constraint into heuristic procedures to obtain a solution to DAD CSP instances. First, we take advantage of the idea that Lagrangian relaxation seeks the greatest lower bound on the original problem. We observe that the Lagrangian goal of maximizing a lower bound is similar to the attacker problem goal of maximizing a shortest path. We combine these ideas into a *Lagrange variable problem* where we treat the Lagrange multiplier as a decision variable to be maximized. We show how regaining the network structure allows us to form a dual ILP version of the operator model under these circumstances. The Dual-ILP Lagrange variable problem then supplants decomposition of the inner AD model in order to find a heuristic defense. This approach is featured in what we call the *phase 1 speed heuristic algorithm*. An advantage of treating the Lagrange multiplier as a decision variable is that it is a very fast way to obtain a heuristic solution. A disadvantage is that it can only produce a lower bound on the optimal solution to an instance. A second method uses Lagrangian relaxation of the operator model follows a more traditional iterative approach to obtain the value of the Lagrange multiplier. In the iterative approach, the upper and lower bounds of the Lagrangian relaxation may not converge, which requires an additional step to find the constrained shortest path of the operator model. We use the technique of path enumeration to close the gap between Lagrangian bounds that do not converge. The second approach is called the *phase 1 bounding heuristic algorithm*. The advantage of the iterative approach is that it can produce valid upper and lower bounds on a DAD CSP instance. A disadvantage is that it is slower than the speed algorithm.

Our two algorithms to obtain heuristic solutions to DAD CSP problem instances can also be incorporated into larger algorithms to obtain optimal or near optimal solutions more quickly than regular nested decomposition. We can use information obtained in the heuristic algorithms as intelligent starting information to begin regular nested

decomposition. The use of smart starting points for nested decomposition serves to dramatically decrease both the time required to obtain a solution and the number of total iterations required within the decomposition procedures. Our most dramatic improvements occur when we combine our phase 1 speed algorithm with nested decomposition in what we call the *phase 2 optimality algorithm*. We show how our combined algorithm is usually more than 100 percent faster than regular nested decomposition in obtaining an optimal solution to DAD CSP instances on a medium-sized grid test network. However, our combined algorithm was found to be marginally slower than nested decomposition in 5 of 49 budget scenarios on the grid network.

We show that our procedures for solving DAD CSP problem instances can be scaled to large real-world problems. We perform an analysis of defending and attacking the shortest distance road paths from the Pentagon to Appomattox, VA subject to a time budget constraint. We show that DAD CSP instances in real-world networks can be very difficult to solve optimally in a reasonable amount of time when attack budgets are greater than one unit. In order to obtain solutions to DAD CSP instances in under 24 hours for our sample problem, we introduce relative tolerances which change our procedures to approximation algorithms. We are able to obtain solutions to DAD CSP on the road network for instances of defense and attack budgets of five units that are guaranteed to be within 10% of optimality in less than 24 hours. Our combined algorithm was faster than regular nested decomposition in 19 of 25 budget scenarios. We show our combined algorithm can obtain a provably near optimal solution from 20% to 6,000% faster than regular nested decomposition on the real world network.

B. FUTURE RESEARCH TOPICS

We believe that many advances can be made for defense and attack of systems by a more robust study of tri-level models that explicitly consider action of the defender, attacker and operation of a system. There are many unexplored avenues for research with the DAD tri-level optimization model.

1. Implicit Enumeration for DAD Constrained Shortest Paths

One future research area is developing an alternative solution method for DAD CSP. We showed how Lagrangian relaxation can be used to solve DAD CSP instances more quickly than regular nested decomposition. However, we believe the IE algorithm could be applied to DAD CSP as another way to solve the problem. Future research can adapt an IE algorithm for the DAD CSP problem as an alternative solution methodology to our Lagrangian relaxation method. It would be interesting to see how the solution performance of IE would fare against regular nested decomposition for DAD CSP problem instances. It might be possible that the bi-level transformation that is performed in IE can provide some benefit to solving DAD CSP more efficiently.

2. Implicit Enumeration Application to Larger or Difficult Problems

Another future research area for the IE algorithm involves scalability limitations of IE to DAD models. We show in Chapter II that IE solutions to the DAD minimum cost flow problem can be slower than regular nested decomposition. We know that solutions to DAD problem instances can become very time consuming when either the defense budget, attack budget or network size become very large. We believe that there may be a tipping point where the analyst can have a preference between the IE and nested decomposition approaches. Is there a problem size limit when solving a DAD instance with IE is impractical? What is the limiting factor in scalability of the IE algorithm, is it network size, attack and/or defense budget size, cardinality of the set of network design elements, or something else?

We also know that IE can provide an alternative method to nested decomposition for solving a DAD model. DAD models may exist that are very difficult to solve with a nested decomposition approach. What are the features of a DAD model that make the nested decomposition method perform poorly? Does an IE approach provide a solution for DAD models when nested decomposition does not?

3. Attack Effects on Side Constraints

A future research area for the DAD CSP problem is attacker influences on in the side constraint. In our exploration of the DAD CSP problem, the attacker is only permitted to change the cost of an arc. This type of attack affects optimality and does not affect the feasible region. Instead, suppose the attacker could attack the side constraint of time. For example, suppose an attacker could damage an arc so that it takes twice as much time to travel across it. This type attack profoundly changes the nature of the problem because the attacker would be able to make some previously feasible paths infeasible. Our current model does not allow for this possibility because our type of attack can only make paths more expensive in terms of distance cost. How does this new kind of attack behavior affect the problem? We believe the changes needed to the problem would be significant. Current nested decomposition methods build both the inner and outer master problem by creating optimality cuts of the feasible region. But, this new type of attack would also introduce the need for additional feasibility cuts that ensure paths containing attacked arcs remain within the feasible region.

4. Determination of Lagrange Multiplier for Outer Decomposition

Another future research area for the DAD CSP problem involves the setting of the Lagrange multiplier in the outer decomposition. In our research, we fix the value of the Lagrange multiplier on the outer decomposition to the value that was found from the inner decomposition. We believe this approach is an acceptable way to quickly find a value for the Lagrange multiplier. Better ways may exist to solve for the value for the Lagrange multiplier in the outer decomposition. Future research could try other methods to set the value of the Lagrange multiplier in the outer decomposition.

5. DAD Modeling for Other Kinds of Systems

Our research into DAD CSP shows that the DAD model can be applied to more than just network flow problems. DAD CSP is an integer linear program that is based on a network, but it does not have network structure because of the side constraint. The next logical step would be to define and formulate tri-level optimization problems that are not

based on networks. Other types of systems that are not networks can be defended against attacks, and that type of analysis could be adapted to the tri-level DAD optimization model.

APPENDIX. ADDITIONAL ALGORITHM PSEUDO-CODE

A. RECURSIVE PATH ENUMERATION ALGORITHM

Recursive Path Enumeration Algorithm Pseudo Code:

Input: Network with nodes and their successors, start_node, end_node, path_cost, path_time, current_path (empty by default), maximum cost allowed, maximum time allowed, attacks to network edges are fixed and the cost of these attacks are incorporated.

Output: The shortest path with the shortest cost and associated time.

Algorithm:

Maximum cost = Lagrangian Upper bound,

Maximum time = time budget.

Current_path = current_path + start node.

If length of current_path > 1 node:

 Calculate path_time and path_cost with data for arc from previous node to start node

Else:

 Set path time = 0 and path cost=0.

If start node is the same as end node:

Return current_path.

Exit function.

Initialize an empty list of paths.

For each current_node that is a successor to the start_node **do**:

If the current_node is not in the path:

If path_time \leq maximum time & path cost \leq maximum cost:

Recursive call to obtain new paths changing these inputs:

 Start_node = current_node.

 Path_time = path_time + arc time from start_node to current_node

 Path_cost = path_cost + arc cost from start_node to current_node

For each new path **do**:

If path_time \leq maximum time & path_cost \leq maximum cost:

 Append the new path to the list of paths.

Sort the list of all time budget feasible paths by their associated costs.

Exit function.

B. AD CSP WITH LAGRANGIAN RELAXATION AND PATH ENUMERATION ALGORITHM

AD CSP with Lagrangian Relaxation & Path Enumeration Algorithm Pseudo-code:

Inputs:

Network with nodes, edges, costs, attack budget and time budget
Maximum iterations for inner AD decomposition (k) & Lagrangian relaxation (L)
Tolerance for AD inner decomposition (usually tolerance ≤ 1 unit)

Outputs:

AD Upper and Lower Bounds, best attack (X^*) and multiplier (μ^*)

Algorithm:

While AD Upper Bound – Lower bound \geq tolerance and iteration \leq max iterations:

For |L| iterations:

 Solve *DAD CSP Operator Model with Lagrangian Relaxation*

 Obtain Lagrangian Upper and Lower bounds

If Lagrangian bound do not match:

 Perform Recursive Path Enumeration

 Obtain shortest path (Y) and associated cost

If Lagrangian bounds match:

 Obtain shortest path (Y) and cost

 Check if shortest path (Y) is repeated from a previous iteration

If path repeated and AD bounds within tolerance:

 Break while and for loops.

If path repeated and AD bounds not within tolerance:

 Resolve *DAD CSP Operator Model with Lagrangian Relaxation*
 with SEC on

 Continue to next iteration

If shortest path cost \geq current AD lower bound:

 Update AD lower bound with shortest path cost

 Save current attack (X) as best attack X^*

 Save current Lagrange multiplier (μ) as best multiplier μ^*

 Solve *AD CSP inner master problem*

 Obtain attack (X) and associated cost

If master problem cost \leq AD upper bound:

 Update AD upper bound

 Increment AD iteration counter

When the inner decomposition upper and lower bounds match, or the iteration limit is reached, then the inner decomposition with Lagrangian relaxation is complete. The output of the inner decomposition is the best attack (X^*) and best Lagrange multiplier (μ^*). Without matching bounds, the output is the best known attack and Lagrange multiplier associated with the final inner decomposition lower bound.

LIST OF REFERENCES

- Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network flows*. Prentice-Hall: Englewood Cliffs, NJ.
- Alderson, D. L., Brown, G. G., & Carlyle, W. M. (2014). Assessing and Improving Operational Resilience of Critical Infrastructures and Other Systems. *In INFORMS Tutorials in Operations Research*. Published online: 27 Oct. 2014, 180–215. Retrieved from <http://dx.doi.org/10.1287/educ.2014.0131>
- Alderson, D. L., Brown, G. G., & Carlyle, W. M. (2015). Operational models of infrastructure resilience. *Risk Analysis*, 35(4), 562–586. Retrieved from <http://onlinelibrary.wiley.com/doi/10.1111/risa.12333/epdf>
- Alderson, D. L., Brown, G. G., Carlyle, W. M., & Cox, L. A. (2013). Sometimes There Is No “Most-Vital” Arc: Assessing and Improving the Operational Resilience of Systems. *Military Operations Research*, 18(1), 21–37. Retrieved from <http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA588521>
- Alderson, D. L., Brown, G. G., Carlyle, W. M., & Wood, R. K. (2011). Solving defender-attacker-defender models for infrastructure defense. *12th INFORMS Computing Society Conference*. Retrieved from <http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA582412>
- Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische mathematik*, 4(1), 238–252. Retrieved from <http://www.im-uff.mat.br/puc-rio/disciplinas/2006.1/soe/arquivos/benders-numerische-mathematik-1962.pdf>
- Bertsimas, D., Tsitsiklis, J. (1997). *Introduction to linear optimization*. Belmont, MA: Athena Scientific.
- Birge, J., Louveaux, F. (2011). *Introduction to Stochastic Programming* (2nd ed.). New York: Springer.
- Brimberg, J., Hansen, P., Lin, K., Mladenović, N., & Breton, M. (2003). An Oil Pipeline Design Problem. *Operations Research* 51(2):228-239. Retrieved from <http://dx.doi.org/10.1287/opre.51.2.228.12786>
- Brown, G., Carlyle, M., Salmerón, J., & Wood, K. (2005). Analyzing the vulnerability of critical infrastructure to attack and planning defenses. *Tutorials in Operations Research: Emerging Theory, Methods, and Applications*, 102–123. <http://www.dtic.mil/dtic/tr/fulltext/u2/a576172.pdf>

- Brown, G., Carlyle, M., Salmerón, J., & Wood, K. (2006). Defending critical infrastructure. *Interfaces*, 36(6), 530–544. Retrieved from <http://www.jstor.org/stable/20141442>
- Carlyle, W. M., Royset, J. O., & Wood, K. R. (2008). Lagrangian relaxation and enumeration for solving constrained shortest-path problems. *Networks*, 52(4), 256–270. Retrieved from <http://onlinelibrary.wiley.com.libproxy.nps.edu/doi/10.1002/net.20247/pdf>
- Carlyle, W. M., & Wood, K. R. (2005). Near-shortest and K-shortest simple paths. *Networks*, 46(2), 98–109. Retrieved from <http://faculty.nps.edu/mcarlyle/docs/CarlyleWoodNSP05.pdf>
- Chartrand, G., & Zhang, P. (2012). *A first course in graph theory*. Mineola, NY: Dover.
- Danskin, J. M. (1967). *The Theory of Max-Min and its Application to Weapons Allocation Problems (Ökonometrie und Unternehmensforschung Econometrics and Operations Research)*. Berlin: Springer-Verlag.
- Ford, L. R., & Fulkerson, D. R. (1962). *Flows in networks*. Princeton, NJ: Princeton University Press.
- Fisher, M. L. (1981). The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1), 1–18. Retrieved from <http://www.jstor.org/stable/30046157>
- Fulkerson, D. R., & Harding, G. C. (1977). Maximizing the minimum source-sink path subject to a budget constraint. *Mathematical Programming* 13(1):116-118. Retrieved from <http://link.springer.com/article/10.1007/BF01584329#>
- GAMS Development Corporation (2016). *GAMS IDE: General Algebraic Modeling System Integrated Development Environment* (Version 24.5) [Software]. Available from <http://www.gams.com>
- Golden, B. (1978). A problem in network interdiction. *Naval Research Logistics Quarterly*, 25(4), 711–713. Retrieved from <http://onlinelibrary.wiley.com/doi/10.1002/nav.3800250412/epdf>
- Google Maps (2016). [Pentagon, Arlington, VA to Appomattox, VA] [Highway Map]. Retrieved from <https://www.google.com/maps/dir/Pentagon,+Arlington,+VA/Appomattox,+VA/@38.0673442,-78.5043271,9z/data=!3m1!4b1!4m13!4m12!1m5!1m1!1s0x89b7b71fabcec10f:0x18d89240e0c6fa0c12m2!1d-77.0537523!2d38.8694499!1m5!1m1!1s0x89b2fb50955a82ab:0xb62498bbe0461e9c!2m2!1d-78.8252911!2d37.3570894>

- Hansen, P., Jaumard, B., & Savard, G. (1992). New branch-and-bound rules for linear bilevel programming. *SIAM Journal on Scientific and Statistical Computing*, 13(5), 1194–1217. Retrieved from <http://epubs.siam.org/doi/abs/10.1137/0913069>
- Harris, T. E., & Ross, F. S. (1955). Fundamentals of a method for evaluating rail net capacities (Working paper No. RM-1573). Retrieved from <http://www.dtic.mil/cgi-bin/GetTRDoc?AD=AD0093458>
- Industrial Business Machines (2015). IBM ILOG CPLEX Solver (Version 12.6) [Software]. Available from <http://www.gams.com>
- INFORMS Computing Society. (2015). In *Mathematical Programming Glossary*. Retrieved from http://glossary.computing.society.informs.org/ver2/mpgwiki/index.php?title=Main_Page
- Israeli, E. (1999). *System interdiction and defense*. Doctoral dissertation. Retrieved from Calhoun <http://calhoun.nps.edu/bitstream/handle/10945/9053/systeminterdicti00isra.pdf?sequence=1&isAllowed=y>
- Israeli, E., & Wood, R. K. (2002). Shortest-path network interdiction. *Networks*, 40(2), 97–111. Retrieved from <http://onlinelibrary.wiley.com/doi/10.1002/net.10039/epdf>
- Koc, A. & Morton, D. P. (2015). Prioritization via stochastic optimization. *Management Science*, 61(3), 586-603. Retrieved from <http://dx.doi.org/10.1287/mnsc.2013.1865>
- LeBlanc, L. J. (1975). An algorithm for the discrete network design problem. *Transportation Science*, 9(3), 183-199. Retrieved from <http://dx.doi.org.libproxy.nps.edu/10.1287/trsc.9.3.183>
- Leon, S. (2010). *Linear algebra with applications* (8th ed.). Upper Saddle River, NJ: Prentice Hall.
- Lim, C. & Smith, J. C. (2007) Algorithms for discrete and continuous multicommodity flow network interdiction problems. *IIE Transactions*, 39(1), 15–26. Retrieved from <http://www.tandfonline.com/doi/pdf/10.1080/07408170600729192>
- Magnanti, T. L. & Wong, R. T. (1984) Network Design and Transportation Planning: Models and Algorithms. *Transportation Science* 18(1):1-55. Retrieved from <http://dx.doi.org/10.1287/trsc.18.1.1>
- McCarl, B. A. (2015) *McCarl expanded GAMS user guide* (version 24.5). Retrieved from <http://www.gams.com>

- McMasters, A. W., & Mustin, T. M. (1970). Optimal interdiction of a supply network. *Naval Research Logistics Quarterly*, 17(3), 261–268. Retrieved from <http://www.dtic.mil/dtic/tr/fulltext/u2/772431.pdf#page=5>
- Morton, D. P., Pan, F., & Saeger, K. J. (2007). Models for nuclear smuggling interdiction. *IIE Transactions*, 39(1), 3–14. Retrieved from <http://www.tandfonline.com/doi/pdf/10.1080/07408170500488956>
- Murray, A. T., & Grubescic, T. H. (2012). Critical infrastructure protection: The vulnerability conundrum. *Telematics and informatics*, 29(1), 56–65. Retrieved from <http://www.sciencedirect.com/science/article/pii/S0736585311000438>
- Nehme, M. V., & Morton, D. P. (2009). Tightening a network interdiction model. In *Proceedings of the 2009 Industrial Engineering Research Conference, Miami, Florida*. Retrieved from http://www.me.utexas.edu/~orie/morton_proceeding/nehme_morton_09.pdf
- Nehme, M. V., & Morton, D. P. (2010). Efficient nested solutions of the bipartite network interdiction problem. In *Proceedings of the 2010 Industrial Engineering Research Conference, Cancun, Mexico*. Retrieved from <http://search.proquest.com.libproxy.nps.edu/docview/733013177/fulltextPDF/8BDFBFD37E984793PQ/1?accountid=12702>
- Owen, G. (1995). *Game Theory* (3rd ed.). Bingley, United Kingdom: Emerald Group.
- Pugh, G. E. (1964) Lagrange Multipliers and the Optimal Allocation of Defense Resources. *Operations Research* 12(4), 543–567. Retrieved from <http://dx.doi.org/10.1287/opre.12.4.543>
- Python Software Foundation. (2003). *Python Patterns—Implementing Graphs*. Retrieved from <https://www.python.org/doc/essays/graphs/>
- Python Software Foundation. (2015). *Python Programming Language* (Version 2.7.10) [Software]. Available at <http://www.python.org>
- Rardin, R. L. (1998). *Optimization in Operations Research*. Upper Saddle River, NJ: Prentice Hall.
- Royset, J. O., Carlyle, W. M., & Wood, R. K. (2009). Routing military aircraft with a constrained shortest-path algorithm. *Military Operations Research*, 14(3), 31–52. Retrieved from <http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA486718>
- Savage, S., Scholtes S., Zwiedler, D. (2006). Probability Management. *OR/MS Today*, 33(1), 20–28. Retrieved from <http://www.lionhrtpub.com/orms/orms-2-06/frprobability.html>

- Scaparra, M. P., & Church, R. L. (2008). A bilevel mixed-integer program for critical infrastructure protection planning. *Computers & Operations Research*, 35(6), 1905–1923. Retrieved from <http://www.sciencedirect.com/science/article/pii/S0305054806002395>
- Smith, J. C., Lim, C., & Sudargho, F. (2007). Survivable network design under optimal and heuristic interdiction scenarios. *Journal of Global Optimization*, 38(2), 181–199. Retrieved from <http://dx.doi.org/10.1007/s10898-006-9067-3>
- Taylor, B. W. (2007). *Introduction to management science* (9th ed.). Upper Saddle River, NJ: Prentice Hall.
- Van Slyke, R. M., & Wets, R. (1969). L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics*, 17(4), 638–663.
- Wagner, H. M. (1975). *Principles of operations research* (2nd ed.). Englewood Cliffs, NJ: Prentice Hall.
- Washburn, A. & Wood, K. (1995). Two-Person Zero-Sum Games for Network Interdiction. *Operations Research* 43(2):243-251. Retrieved from <http://dx.doi.org/10.1287/opre.43.2.243>.
- Wollmer, R. (1964) Removing Arcs from a Network. *Operations Research* 12(6):934-940. Retrieved from <http://dx.doi.org/10.1287/opre.12.6.934>
- Wolsey, L. A. (1998). *Integer Programming*. Hoboken, NJ: John Wiley & Sons.
- Wood, R. K. (1993). Deterministic network interdiction. *Mathematical and Computer Modelling*, 17(2), 1–18. Retrieved from <http://www.sciencedirect.com/science/article/pii/089571779390236R>

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California